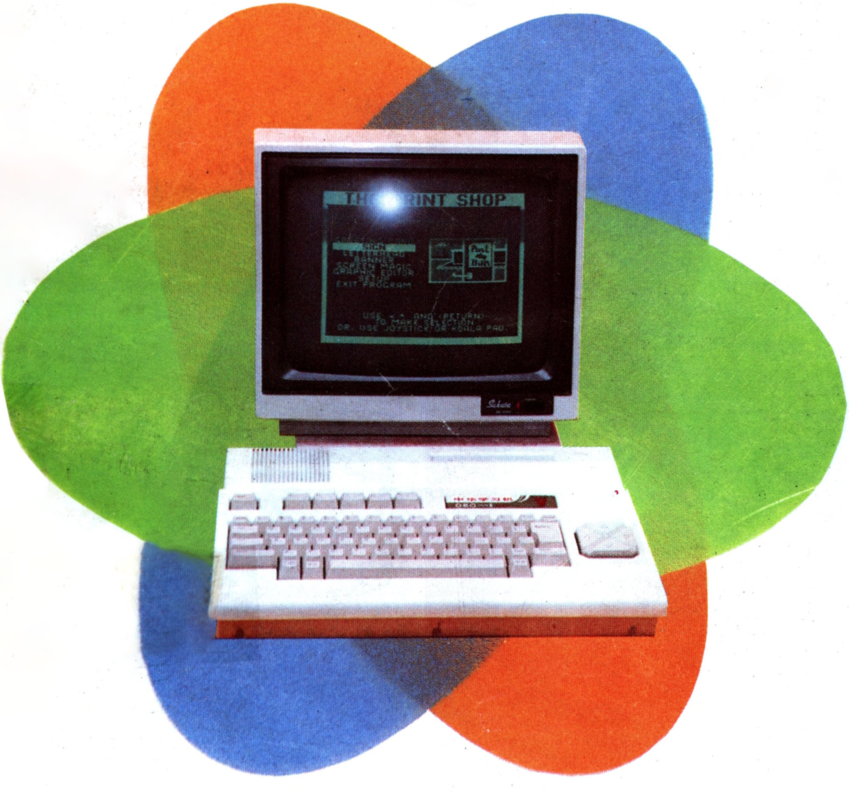
中华学习机实用大全 ④

FORTRAN与Pascal语言

韩仲清　主编



電子工業出版社

书号 C-666
账号
年 月 日

中华学习机实用大全④

FORTRAN与Pascal语言

韩仲清 主编

工厂资料室
館藏

电子工业出版社

内 容 提 要

本书从实用的角度出发，详细介绍了中华学习机CEC-I的FORTRAN和PASCAL语言。主要内容包括：FORTRAN语言的基本概念，简单程序、分支程序、循环程序、数组、函数和子程序以及文件系统。PASCAL语言基础，纯量、集合、数组、记录、文件和指针，过程和函数，PASCAL程序的编译和连接。

本书的最大特点是内容充实、具体、实用、易学。最适宜于广大青少年、中小学生及其家长和计算机爱好者自学，也可以作为高等院校非计算机专业、培训班、函授班、职业学校、中等专业学校学生学习计算机课程的教材 还可供从事计算机研究和应用的人员使用。

中华学习机实用大全 ④

FORTRAN与Pascal语言

韩仲清 主编

责任编辑 王昌铭

电子工业出版社出版（北京市万寿路）

电子工业出版社发行 各地新华书店经销

中国科学院印刷厂印刷

开本：787×1092毫米1/32 印张：9.75 字数：227千字

1990年9月第1版 1990年9月第1次印刷

印数： 10700 册 定价：3.80元

ISBN7-5053-1108-5/TP·178

前　言

中华学习机以前所未有的速度进入寻常人家，成为人们工作、学习和生活的得力助手，尤其是在开发青少年的智力方面，已经显示出了强大的威力。

为满足广大青少年、中小学生、家长和计算机爱好者对中华学习机知识的渴求，我们组撰了这套《中华学习机实用大全》。该书内容丰富、具体、实用；把中华学习机的最新软件以及最实用、最急需的技术、技巧和方法毫无保留地介绍给读者，使初学者很快入门，入门者进一步提高；学到知识，掌握技术，增长才干，启迪智慧，得到力量，增强解决实际问题的能力。

《中华学习机实用大全》分为七册：

1.BASIC与LOGO语言

2.汉字处理与数据库技术

3.操作系统

4.FORTRAN与PASCAL语言

5.汇编语言程序设计

6.游戏与绘图

7.硬件维修与经验技巧

为便于阅读和使用，每册内容彼此均是独立的，读者可以从任何一本书开始阅读。但是，如果读者是计算机技术的初学者，那么最好按顺序阅读。当然，每本书中可以只选学自己感兴趣的那部分内容。

《中华学习机实用大全》在内容安排上，由浅入深，循序渐进。既考虑到初学者很快入门，又考虑到让入门者进一步提高，还考虑了应用者能够实用。书中有较多实例，读者可以边读、边学、边用、边想、边写(写自己的程序)。在结构安排上，既便于自学，又可以作为教材。在文字叙述上，力求浅显、通俗、易懂。在选材上，突出实用性技术。

本书是《中华学习机实用大全》的第四册，主要介绍了FORTRAN和PASCAL语言的基本概念；编写程序的方法以及如何上机操作。虽然在IBM PC机上FORTRAN和PASCAL已是累见不鲜，但在中华学习机上使用这两种语言，本书的介绍却是第一次。读者不妨参照书本上机试一试。

欢迎读者对本书进行品评，指出疏漏和错误，我们将甚为感谢！

在编写本书的过程中，电子工业出版社和电子报社的编辑们给予了指导和帮助，特别是王有春、孙萌老师提出了许多宝贵的修改意见；为调试和运行示例程序，成都三开元电脑部经理舒新生无偿地提供了CEC-I中华学习机及其软件；张陛楷副教授和何晓林审阅了全部书稿。在此一并表示感谢！

参加本书编写的有：韩仲清，黄金姬，揭金良，廖兴祥，刘元社。全书由韩仲清统稿。

编　者

1989年11月14日于四川大学

目　　录

第一部分　FORTRAN语言程序设计

第一章　FORTRAN的基本知识 ……………………………（ 2 ）

1.1　程序结构 ……………………………………………（ 2 ）

1.1.1　源程序的结构 ……………………………………（ 2 ）

1.1.2　FORTRAN字符集 ………………………………（ 5 ）

1.1.3　程序行的格式 ……………………………………（ 6 ）

1.2　数据类型和常数 ……………………………………（ 7 ）

1.3　FORTRAN名 ………………………………………（ 9 ）

1.4　算术表达式 …………………………………………（ 13 ）

1.5　内部函数简介 ………………………………………（ 16 ）

习题一 ……………………………………………………（ 17 ）

第二章　简单程序……………………………………………（ 19 ）

2.1　赋值语句 ……………………………………………（ 19 ）

2.2　数据语句 ……………………………………………（ 21 ）

2.3　简单的输入输出 ……………………………………（ 23 ）

2.3.1　输入输出语句 ……………………………………（ 23 ）

2.3.2　格式说明及其应用 ………………………………（ 25 ）

习题二 ……………………………………………………（ 33 ）

第三章　FORTRAN程序的运行 ……………………………（ 35 ）

3.1　UCSD PASCAL下的FORTRAN配置 ………………（ 35 ）

3.2　CEC-I上的FORTRAN配置 …………………………（ 37 ）

3.2.1　CEC-I需要哪些系统文件 ………………………（ 37 ）

3.2.2 如何构造CEC-I的系统文件盘 …………………… (38)
3.3 FORTRAN 77程序的上机过程 …………………… (39)
3.3.1 启动UCSD操作系统 …………………… (39)
3.3.2 源程序的输入和修改 …………………… (41)
3.3.3 编译源程序 …………………… (46)
3.3 4 连接P代码文件 …………………… (47)
3.3.5 执行一个用户程序 …………………… (49)
习题 三 …………………… (50)
第四章 控制语句及分支程序 …………………… (51)
4.1 转移语句 …………………… (51)
4.2 IF语句 …………………… (57)
4.2.1 算术 IF 语句 …………………… (57)
4.2.2 逻辑 IF 语句 …………………… (59)
4.3 停语句 …………………… (66)
4.4 判定结构程序 …………………… (67)
习 题 四 …………………… (76)
第五章 循环程序设计 …………………… (78)
5.1 循环程序的概念 …………………… (78)
5.2 循环语句 …………………… (80)
5.3 继续语句 …………………… (86)
5.4 循环程序的嵌套结构 …………………… (86)
习 题 五 …………………… (93)
第六章 数组 …………………… (95)
6.1 数组的概念 …………………… (95)
6.2 数组的定义 …………………… (97)
6.2.1 数组说明语句 …………………… (97)
6.2.2 用类型语句定义数组 …………………… (99)
6.3 数组的存储与输入输出 …………………… (100)
6.3.1 数组元素的存储顺序 …………………… (100)

6.3.2 数组的输入输出 …………………………………………（102）
6.4 数组的应用实例 …………………………………………（107）
习题六 …………………………………………（109）
第七章 函数与子程序 …………………………………………（111）
7.1 函数 …………………………………………（111）
7.1.1 内部函数 …………………………………………（111）
7.1.2 语句函数 …………………………………………（112）
7.2 函数子程序 …………………………………………（114）
7.2.1 函数子程序的定义 …………………………………………（115）
7.2.2 函数子程序的引用形式 …………………………………………（117）
7.3 子例程子程序 …………………………………………（125）
7.3.1 子例程子程序的结构 …………………………………………（126）
7.3.2 可调数组与假定大小数组 …………………………………………（130）
7.4 等价语句与公用语句 …………………………………………（133）
7.4.1 等价语句 …………………………………………（133）
7.4.2 公用语句 …………………………………………（135）
习题七 …………………………………………（140）
第八章 文件及输入输出系统 …………………………………………（142）
8.1 文件的基本概念 …………………………………………（142）
8.2 文件的打开与关闭 …………………………………………（145）
8.3 其它编辑描述符 …………………………………………（148）
8.4 文件的输入输出 …………………………………………（152）
8.4.1 输入输出语句 …………………………………………（152）
8.4.2 顺序文件的建立和检索 …………………………………………（154）
8.4.3 直接文件的建立和检索 …………………………………………（156）
8.4.4 格式文件的建立和修改 …………………………………………（160）
8.4.5 无格式文件的建立和检索 …………………………………………（163）
8.4.6 外部文件和内部文件 …………………………………………（166）
8.5 文件定位语句 …………………………………………（167）

第二部分　PASCAL语言程序设计

第九章　PASCAL语言基础 ……………………………………（172）
9.1　语言的基本元素 ……………………………………（172）
9.2　程序的结构 ……………………………………（173）
9.3　标准纯量类型 ……………………………………（176）
9.4　PASCAL的通用语句 ……………………………………（178）
9.4.1　基本语句 ……………………………………（178）
9.4.2　构造型语句 ……………………………………（182）
习题九 ……………………………………（191）
第十章　PASCAL程序的运行 ……………………………………（193）
10.1　PASCAL系统的启动 ……………………………………（193）
10.2　磁盘的格式化和复制 ……………………………………（194）
10.3　运行PASCAL程序 ……………………………………（197）
10.3.1　PASCAL程序的输入 ……………………………………（197）
10.3.2　修改程序 ……………………………………（199）
10.3.3　编译和运行程序 ……………………………………（201）
10.4　工作文件的更新 ……………………………………（203）
习题十 ……………………………………（205）
第十一章　纯量、子界、集合、数组和串类型 ………（206）
11.1　纯量类型 ……………………………………（206）
11.2　子界类型 ……………………………………（209）
11.3　长整数类型 ……………………………………（211）
11.4　集合类型 ……………………………………（213）
11.5　数组类型 ……………………………………（218）
11.6　串类型 ……………………………………（221）
11.6.1　串变量及其赋值 ……………………………………（221）
11.6.2　串标准函数和过程 ……………………………………（224）

习题十一 …………………………………………………………（227）
第十二章　记录、文件和指针类型 ……………………………（231）
12.1　记录类型 ……………………………………………………（231）
12.2　文件类型 ……………………………………………………（235）
12.2.1　文件类型定义 ……………………………………………（236）
12.2.2　文件操作 …………………………………………………（236）
12.2.3　类型文件 …………………………………………………（239）
12.2.4　文本文件和交互文件……………………………………（243）
12.2.5　无类型文件 ………………………………………………（250）
12.2.6　预定义文件 ………………………………………………（252）
12.2.7　输入输出检查 ……………………………………………（252）
12.3　指针类型………………………………………………………（254）
习题十二 …………………………………………………………（260）
第十三章　过程和函数 …………………………………………（262）
13.1　过程的定义和调用……………………………………………（262）
13.2　函数的定义和调用……………………………………………（266）
13.3　标准过程和标准函数…………………………………………（267）
13.3.1　处理字节的过程和函数 …………………………………（268）
13.3.2　与设备有关的过程和函数 ………………………………（271）
13.3.3　超越函数 …………………………………………………（273）
13.3.4　其它过程和函数 …………………………………………（274）
习题十三……………………………………………………………（277）
附录一　FORTRAN语言一览表 ………………………………（278）
1.1　FORTRAN语句一览表 ………………………………………（278）
1.2　内部函数表 ……………………………………………………（281）
1.3　FORTRAN出错信息 …………………………………………（283）
附录二　PASCAL语言一览表 …………………………………（292）
2.1　PASCAL命令一览表 …………………………………………（292）
2.2　PASCAL出错信息表 …………………………………………（293）

第一部分
FORTRAN语言程序设计

本部分介绍了中华学习机CEC-I FORTRAN77语言的基本概念和程序设计方法。主要包括：FORTRAN 的基本知识；简单程序；FORTRAN程序的运行；控制语句及分支程序；循环程序；数组；函数与子程序；文件及输入输出系统。为写起来简洁，这里将中华学习机 CEC-I 上的FORTRAN77语言简称为FORTRAN。

第一章　FORTRAN的基本知识

中华学习机上的FORTRAN语言是ANSI FORTRAN 77的一个子集，它与APPLE FORTRAN基本兼容。在CEC-I上能够运行的FORTRAN程序，在APPLE微机上也能运行。由于FORTRAN可以产生覆盖程序段，因而在CEC-I机上也能运行较大的程序。

1.1　程序结构

用FORTRAN语言编制的程序称为源程序。源程序是由一些程序单位组成的，程序单位又由语句组成，每个语句又由规定的字母、数字等符号组成。要使用FORTRAN语言，首先得了解它的基本组成部分以及程序的结构特征。

1.1.1　源程序的结构

一个FORTRAN源程序是由一个或多个程序单位组成的。这些程序单位可以是主程序、子程序、函数子程序等。其形式如下：

第1程序单位 {
〈程序单位首部〉
〈程序段体〉
END

第2程序单位 {
〈程序单位首部〉
〈程序段体〉
END

⋮

1.程序单位首部

程序单位首部标志着一个程序单位的开始，它说明程序单位的属性，提供必需的参数以及参数说明等信息。

程序单位首部指的是主程序、子例程子程序和函数子程序首部语句。

（1）主程序首部语句

主程序是以PROGRAM语句开始的。主程序语句开头的程序单位即是主程序单位，主程序单位的PROGRAM语句可以省略，这时系统自动按主程序处理。主程序语句格式为：

PROGRAM〈程序名〉

在一个可执行的FORTRAN程序中只允许有一个主程序。

（2）子例程子程序首部语句

在一个FORTRAN程序中可以出现一个或多个子例程子程序。子例程子程序首部语句的格式为：

SUBROUTINE〈子程序名〉〔（〈形参表〉）〕

其中SUBROUTINE为子程序单位的定义标志，〈形参表〉为子程序中所要用到的形式参数表。如果不带形式参数，则称为无参子程序。（方括号〔…〕中的内容根据需要而定，称为待选择项，下同）

（3）函数子程序首部语句

函数子程序首部语句为：

〔〈类型〉〕FUNCTION〈函数名〉（〔〈形参表〉〕）

函数子程序首部用于定义不同类型的函数，除必须指明类型外，其它与子程序类同。

2.程序段体

程序段体是程序单位的主要部分。它是由一个或多个FORTRAN语句组成的实体，以实现整个处理过程。这个语句实体主要包括可执行语句和非执行语句两大类。

（1）可执行语句

可执行语句指明动作并在可执行程序中形成执行序列，如获得各种计算值、程序的控制转移、数据的传输、子程序的调用等。可执行语句包括赋值语句、控制语句和I/O语句。

（2）非执行语句

非执行语句用于指明数据的特性、排列和初值，包括输入输出编辑信息，语句函数。非执行语句不是执行序列的一部分。非执行语句可以带标号，但这种语句标号不用于控制转移。

3.程序单位结束语句

在FORTRAN语言程序中，用END语句来结束一个程序单位。

当END出现在主程序中时，它结束整个可执行程序的执行。当END出现在子程序或函数子程序中时，其作用相当于RETURN（返回）语句。记住：FORTRAN程序的每一个程序单位的最末一行必须而且只能是END语句。

4.程序单位中语句的顺序

程序单位中的语句要依一定次序排列，各种语句出现的先后顺序如下：

<table>
<tr><td rowspan="6">注释行</td><td colspan="3">PROGRAM, FUNCTION 或 SUBROUTINE 语句</td></tr>
<tr><td rowspan="5">编译控制语句</td><td rowspan="5">FORMAT 语句</td><td>IMPLICIT 语句</td></tr>
<tr><td>其它说明语句</td></tr>
<tr><td>DATA 语句</td></tr>
<tr><td>语句函数语句</td></tr>
<tr><td>可执行语句</td></tr>
<tr><td colspan="4">END 语句</td></tr>
<tr><td colspan="4">空格行</td></tr>
</table>

1.1.2 FORTRAN 字符集

FORTRAN字符集由26个英文字母，10个数字和13个专用字符组成。它们是：

1.字母

ABCDEFGHIJKLMNOPQRSTUVWXYZ

2.数字

0 1 2 3 4 5 6 7 8 9

3.专用字符

字 符	字符名	字 符	字符名
␣	空白(空格)	)	右括弧
=	等 号	,	逗 号
+	加 号	.	圆 点
−	减 号	$	美元符号
*	星 号	'	单撇号

/	斜 线	:	冒 号
(	左括弧		

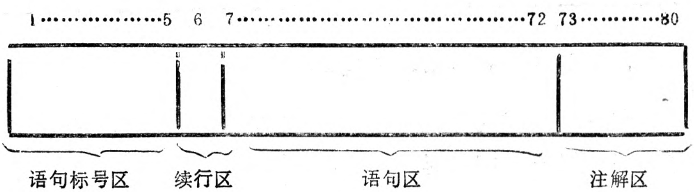
注意：

（1）空白符在显示器和打印机上输出时均是一个空白位置。为了明显起见，有时用“␣”表示一个空白。在程序中也可以用连续的空白符对源程序分段。

（2）书写程序时，大小写字母均可。但在UCSD操作系统下均为大写字母。

1.1.3 程序行的格式

FORTRAN语言的程序行，实际就是一个语句行。每一个语句行均由三部分组成。其形式为：



1.语句标号区

第1～5列称为标号区，它由至多5位无符号数字组成。标号中的空格符不起作用，例如语句标号“1␣43”实际是“143”。语句标号可根据需要而设置，如果不需要语句标号，标号区也必须空出来，不允许有任何符号或全零。在同一个程序单位中不允许出现完全相同的语句标号。

语句标号必须出现在语句的起始行上。

如果语句标号的第一列为“C”或“*”字符，则表示该语句行为注解行。注解行为非执行语句行，它可以在程序单位中的任意一行出现。FORTRAN编译程序对注解行不予

处理，对程序的执行毫无影响，只对程序起解释说明的作用。

注解行的书写可不分区而连续书写，直到本行结束。如果一个语句行写不完，可在下一行继续书写，但第一列必须是“*”或“C”。

2.续行区

第6列称为续行区。当一个语句在一行内写不完时，可在下一行的语句区接着书写，但必须用非零或非空格的任一个FORTRAN字符在续行区上作标记。这些行中的第一行称为起始行，其余为继续行。起始行的第6列必须是空格或零，只有起始行才有标号，续行的标号区必须全为空白，不允许带有语句的标号。续行最多不超过9行。

3.语句区

第7～72列称为语句区。所有语句都必须写在语句区。当一行写不完时，可用续行标志，在下一行的语句区中继续书写。

综上所述，一个语句最多占用10行（即第一行为该语句的起始行，其余9行为续行），从第7列到第72列的字符总数为660个。

1.2 数据类型和常数

FORTRAN语言中的数据有四种基本类型：整型（INTEGER）、实型（REAL）、逻辑型（LOGICAL）和字符型（CHARACTER）。

1.整型常数

整型常数是没有小数点的整数，它由0～9的数字组成，

当然，可以有正、负号。整型常数的值在计算机内存中的表示是精确的，通常占用2个字节的存储单元。

整型常数可以是－32768～32767之间的任一值。如：

1567，2568，－576，＋3780，0，005607等，而2.7867，10^{-2}都不是整型常数。注意：整型常数之前的零可以被忽略，正整常数之前的正号"＋"可以省写。

2.实型常数

实型常数分为两种形式：

（1）基本实型常数，它由整数部分、小数点和小数部分组成。可以只有整数或小数部分，但不能两者同时都没有。如：

3.14159，－327.54，＋5.327，100.，.532都是正确的基本实型常数。

（2）基本实型常数后跟指数部分（以字母E标识），如：15.261E5，－3.4E－3，1E5等。

实型常数在内存中占用4个字节，其精度为6位十进制数字。实型数的表示范围是：

－1.70141E＋38～－5.87747E－39（负数范围）

5.87747E－39～1.70141E＋38　（正数范围）

3.逻辑型常数

逻辑型常数只有"真"和"假"两个值，占用2个字节的存储容量。用·TRUE·表示"真"值，用·FALSE·表示"假"值。

在计算机内存中，·FALSE·用0表示，·TRUE·用1表示。

4.字符型常数

字符型常数由一串ASCII码字符组成，串中字符的个数

称为字符长度。串中的每一个字符占用1个字节的存储空间。

字符型常数的表示形式是由两个单撇号将字符串括起来，撇号不算作字符常数的一部分。若字符串内出现撇号，则可用两个相邻的撇号来表示，但只计作一个撇号。如：

'CEC－FORTRAN77'，其值为：CEC－FORTRAN 77

'AB␣␣CD'，其值为：AB␣␣CD

1.3 FORTRAN名

FORTRAN名表示用户或系统定义的变量、数组、函数或程序单位名，在后面的章节里将逐一给予介绍。

1.变量

变量是指在程序执行过程中其值可以变化的量，例如：

X＝3.0 （这是赋值语句，下同）

X＝5.0，这里的X就是变量。

在执行第一个语句时，X的值为3.0，执行第二个语句后X的值变成了5.0。变量需要在程序中通过某些途径予以赋值。

2.变量名

每一个变量都需要有一个名字。但在同一程序单位内不同的变量不能同名。

在FORTRAN程序中，空格符被忽略不计。因此，变量名中有空格是允许的（这不同于dBASE的文件名和字段名）。

FORTRAN语言规定，变量名必须以字母开头，其后

最多跟5个字母或数字（空格符不计算在内）。

下面的变量名是正确的：

ABC，XY1，Y1，Y，ABCDE，A␣B5D

下面的变量名是错误的：

A.FOR（小数点不能出现在变量名中）

2AX（不是字母开头）

YX－12（减号不允许）

注意：

（1）FORTRAN变量名可选用任何允许的字符序列。但是为了避免与数学库内某些名字发生冲突，造成连接错误。最好不用系统关键字（如：IF、THEN、ELSE等）、数学库内的名字或内部函数名作变量名。

（2）变量名应尽量采用与数学上或实际问题中相同或相近的名字，直观易懂。

（3）本语言标准规定：变量名不超过6个字符，超过则作出错处理，

3.FORTRAN名的作用域

一个FORTRAN名的作用域是指这个名字在哪个区域内被引用才是有效的。通常，一个FORTRAN名的作用域是一个可执行的程序、一个程序单位或一个函数语句。主程序名、函数子程序名、子程序名、公用块名的作用域是整个可执行的程序。这样的FORTRAN名称为全局名，其作用域称为全局作用域；变量名、数组、常数、语句函数、形式参数等名的作用域是一个程序单位，这种FORTRAN名称为局部名。函数语句中出现的参数名的作用域是该语句函数，在该语句函数内不作他用，在语句函数以外可作别的用途。

4.变量的类型及说明

变量也和常量一样具有四种数据类型：整型、实型、逻辑型以及字符型。下面先介绍整型（INTEGER）和实型（REAL）以及隐含说明。

（1）隐式说明

如果变量名的第1个字母是I、J、K、L、M或N之一，那么这种变量名看成是整型变量名，所有其它字母打头的变量名均为实型变量名。我们称这种规则为“I-N”规则。如：

MAX，MIN，IABC，JALFA 隐含说明为整型变量

AMAX，BMIN，C1，C2，XYZ 隐含说明为实型变量

（2）显式说明

如果程序中用INTEGER或REAL明确地指定了变量所具有的类型，那么这种情况称为显式说明。

语句格式为：

INTEGER〈变量名$_1$〉〔，〈变量名$_2$〉〕…

REAL〈变量名$_3$〉〔，〈变量名$_4$〉〕…

其中INTEGER、REAL分别是整型、实型类型说明符。这就指明变量名1、变量名2为整型变量，变量名3、变量名4为实型变量，如：

INTEGER A，B，C

REAL X，Y，Z

整型变量在内存中分配相邻的2个字节的存储单元。实型变量在内存分配相邻的4个字节的存储单元。

一个说明语句可以给若干个变量指定同样的数据类型，但各变量名之间必须用逗号隔开。在同一个程序单位中，变量名只能在类型说明语句中出现一次。同一类型的说明语句

可以出现多个，顺序不限。

除用INTEGER和REAL语句分别定义整型和实型变量外，还可用CHARACTER语句定义字符类型的变量。其一般格式是：

CHARACTER 〔*n〔，〕〕〈变量名1〉〔*n〕〔,〈变量名2〉〔*n〕〕…其中：CHARACTER是字符类型定义符；变量名1、变量名2均可以是一个变量名、数组名或数组说明符；n是字符变量或字符数组元素的字符个数（或长度），它必须是一个1到255之间的无符号整数。若省略*n，则字符元素的长度为1。

下面的字符说明语句是正确的：

CHARACTER *50，B（B是长度为50的字符变量）

CHARACTER C*50，CHR（C的长度为50,CHR的长度为1）

如果在CHARACTER语句中出现数组名或数组定义符，则*n指明每一个数组元素的长度。

注意：若长度附于CHARACTER后面，则说明其后出现的字符变量的长度均为n。若在变量名后附上的长度说明为*n，，则该变量的长度以此为准。例如：

CHARACTER*20 A，B*20相当于：

CHARACTER*20 A，B或

CHARACTER A*20，B*20

（3）隐含说明语句

隐含说明语句用于确定用户的变量的隐含规则。格式为：

IMPLICIT TYPE（a〔，a〕…）

其中：IMPLICIT为隐含说明语句的定义符。TYPE为变量

的数据类型，它可以是INTEGER、REAL、LOGICAL以及CHARACTER四种。a代表同类变量名的第一个字母，即指定了以某个字母开头的那类变量的数据类型。如：

IMPLICIT INTEGER（A，B），REAL（M，N）

这说明以A或B打头的变量名均为整型。以M、N打头的变量名均为实型。

用这种方式说明以后，可以不再用显式方式语句再进行说明。即使说明了，也只是对其中某一个变量而言，而不影响这一群变量的类型，这就是IMPLICIT语句和其他类型说明语句的区别。

IMPLICIT除了用单个字母来指定一群变量的数据类型以外，还可以用字母范围来指定变量的类型。例如：

IMPLICIT INTEGER（P—T），REAL（J—O）

这表示以P、Q、R、S、T中任一个字母开头的变量名均为整型，而以J、K、L、M、N、O开头的变量名均为实型。

IMPLICIT语句中没有定义的变量，仍遵循I-N规则。若有其它显式说明，则以显示说明为准。

在一个程序单位中，IMPLICIT语句必须位于所有其它说明语句之前。在IMPLICIT语句中的字母也只能出现一次。

1.4 算术表达式

1.算术运算符

FORTRAN的算术运算符有：

运算符	二元运算符	一元运算符
**	乘方（幂）	
/	除法	
*	乘法	
+	加法	正
−	减法	负

注意：FORTRAN语言中用星号"*"表示数学中的乘号"×"，用"**"来表示乘方。所谓二元运算是指一个运算符具有两个运算对象；一元运算是指一个运算符只有一个运算对象。

2.什么是算术表达式

算术表达式是用算术运算符将数值常数、数值变量和数值函数按一定规则连接起来的产生数值的式子。当然，可以成对地使用圆括号，如：

数学公式	算术表达式
πr^2	3.1415926*R**2
$\frac{a-b}{a+b}$	(A−B)/(A+B)
$\sqrt{b^2-4ac}$	SQRT(B*B−4.*A*B)
$\mathrm{Sin}^2x-\mathrm{Cos}^2x$	SIN(X)**2−COS(X**2)

算术表达式的书写是否正确，是算术表达式计算结果正确与否的关键。因此要注意：乘号不能省略或省写，适当地成对使用圆括号。

3.算术运算符的优先级

FORTRAN语言中算术运算符的优先级按由高到低是：

* *、*或/、+或-。同级运算从左到右，若有括号则优先。

在计算算术表达式时，系统按照运算符的优先级别进行处理。

如果算术表达式中有括号以及函数引用，则求值步骤如下：

（1）如果表达式中有括号，则首先计算。如果有多层括号嵌套，则从最内层开始计算，然后逐步向外扩展，直至最外一层括号。

（2）如果有函数引用，则首先进行函数计算。

4.算术表达式的类型

算术表达式象变量、常量一样具有数据类型，从根本上说算术表达式的数据类型是由算术表达式的运算对象的类型所决定的。当算术表达式中的运算对象的类型不一致时，则系统将它们转换成同一类型后进行运算，该类型即为算术表达式的数据类型。

数据类型的转换规则是：

（1）整型算术表达式

整型算术表达式由整型运算对象组成，计算的结果为整型值。

注意：在作整数除法运算时，运算的结果只取商的整数部分，小数部分被丢掉。

（2）实型表达式

实型算术表达式由实型运算对象组成，计算的结果为实型值。

（3）混合型算术表达式

混合型算术表达式是由整型、实型运算对象组成的，计算的结果是实型。实际上，在进行运算时将运算符左右两端的

运算对象的数据类型由整型转换成实型后再计算。

需要指出的是：数据类型的转换不是一次进行的，而是逐步进行的。即一边转换一边计算。

若整、实型常数超过规定的上下限，则称为“溢出”，超过上限叫做“上溢出”。超过下限叫“下溢出”。若出现上溢出，则计算机立即停止程序的运行，等待用户进行处理。对于下溢出则视计算机系统而定，有的当作错误处理，而大多数的计算机系统将这种情况的数据当作零处理。

另外，在计算机数据处理中实数的值是一个近似值，其精确度的高低取决于实数的有效数字位数，标准实数的有效数字为6位。

1.5 内部函数简介

在数值计算过程中，经常要用到一些计算公式或计算过程，如果每个用户都去编写这些程序，则给用户带来麻烦。为了提高编程序的效率，FORTRAN系统提供了一系列定义好的函数，它们具有专门的意义，用户在编写程序时，可在表达式中直接引用这些函数（表1-1）。

表1-1 常用内部函数

函数	功能	自变量类型	函数值类型
SIN(X)	求自变量的正弦值	实	实
COS(X)	求自变量的余弦值	实	实
TAN(X)	求自变量的正切值	实	实
ASIN(X)	求自变量的反正弦值 $\mid$自变量$\mid \leqslant 1$，$\mid$结果$\mid \leqslant \frac{\pi}{2}$	实	实

续表1-1

函数	功能	自变量类型	函数值类型
ACOS(X)	求自变量的反余弦值。 |自变量|≤1，0≤结果≤π	实	实
ATAN(X)	求自变量的反正切值，|结果|≤$\frac{\pi}{2}$	实	实
ATAN2(X,Y)	求两个自变量的反正切值。|结果|≤π	实	实
SQRT(X)	求自变量的平方根。X≥0	实	实
EXP(X)	求自变量以e为底的指数函数值	实	实
ALOG(X)	求自变量以e为底的自然对数值	实	实
ALOG10(X)	求自变量以10为底的对数值	实	实
ABS(X)	求自变量的绝对值	实	实
INT(X)	取自变量的整数部分	实	整
REAL(X)	将自变量的值转为实型值	整	实
AINT(X)	取自变量的整数部分为实型值	实	实
NINT(X)	取自变量舍入后的整型值	实	整
ANINT(X)	取自变量舍入后的实型值	实	实

习题一

1.下面的数哪些是整数？哪些是实数？

35，27.68，56.，1250，500E−3，1.25E2

2.把下列数据表示为FORTRAN语言所允许的数据。

10^{-8}，12×10^{2}，0.45×10^{0}，567.0，327678

56%，$\frac{1}{1000}$，$\frac{1}{10^{-2}}$，0.000478

3.下列符号串哪些是FORTRAN名？哪些不是？为什么？

ABC，A12，1AB，X␣YZ，ABCDEFGH，

α，βAB，C-567，$78A，A #4

4.用FORTRAN表达式来表示下列式子。

$-(x^2+y^2)$, $\dfrac{1}{1+\dfrac{1}{1+\dfrac{1}{x+1}}}$, $\left(\dfrac{y}{x}\right)^k$, $(-k)^r$

$|x-y|<10^{-2}$, $\cos\alpha+\sin\beta$, $\sqrt{x}+y^2+e^x$

第二章　简单程序

一个FORTRAN程序由一个系列的语句组成。学习编写FORTRAN程序，首先就要了解这些语句，它们各自有什么作用，如何书写，在程序中所处的位置等等。

我们这里所说的简单程序指的是由一个FORTRAN主程序单位构成的程序。先看一个例子。

例1　求$f(x)=ax^2+bx+c$在给定a，b，c及x后的函数值。

```
A=3.0
B=5.0
C=-2.0
X=2.0
FX=A*X*X+B*X+C
WRITE(*,'(1X,''F(X)='',F6.2)') FX
END
```

这就是一个FORTRAN程序。其中：第一个语句到第四个语句为算术赋值语句，它们将实型常数3.0、5.0、－2.0和2.0分别赋给实型变量A、B、C和X。第五个语句也是赋值语句，它将计算结果送到实型变量FX中。第六句是输出语句，它将计算结果显示出来。最后是程序结束语句。下面我们来介绍这些语句。

2.1　数值语句

1.算术赋值语句

算术赋值语句可用来确定整、实型变量的值，它是数值计算中主要使用的语句之一。

格式：〈变量名〉=〈算术表达式〉

作用：将算术表达式的值送到左部变量中。

一般要求左部变量和右端表达式的数据类型必须一致。如果数据类型不一致，赋值语句则以左部变量的数据类型为准，将算术表达式的值转换成左部变量的类型后再进行传送。

下面是算术赋值语句的例子：

```
FX=A*X*X+B*X+C
AB=1.56
I=2
L=L+1
```

例中L=L+1是将L的当前值加上整数1后再赋给整型变量L。

下面的赋值语句是不正确的：

```
A=B=X+3.14159
X+1=SQRT(B**2-4*A*C)
```

在FORTRAN语言中，一个赋值语句只能对一个变量赋值，所以A=B=X+3.14159是错误的。如果要对不同的变量赋同一个表达式的值，则必须分开写成多个赋值语句。如：

```
A=X+3.14159
B=X+3.14159
```

另一个赋值语句的错误原因请读者自己考虑。

2.字符赋值语句

前面介绍的算术赋值语句，只能对数值型量进行赋值，

对于字符型量的赋值则要通过字符赋值语句来实现。

格式：〈变量名〉=〈字符表达式〉

其中，左端变量名只能是字符型变量名，即必须在字符说明语句CHARACTER中加以定义。

右端为字符表达式，它产生一个字符型值。字符表达式可以是字符常量、字符变量、字符型数组元素。

2.2 数据语句

算术赋值语句可以将常数赋给左部变量，实现为变量赋初值的目的。如果数据很多，要用这种方式赋初值，程序就写得太长了，不大适合实际工作的需要。是否还可以用其它语句来对变量赋初值呢？当然可以。这就是DATA数据语句的作用。

格式：DATA 〈变量表〉/〈常数表〉/

其中：变量表可以是一个或多个整型、实型变量名、字符变量名、数组名或数组元素名，它们彼此之间用逗号“，”隔开。

常数表中的常数可以是整型、实型或字符型常数，常数的类型必须与所对应的变量名的数据类型相同。常数之间也用逗号“，”分隔，并且括在两条斜线“/”之间。常数的顺序应与变量表中变量的顺序、个数一一对应，如：

DATA X，Y，Z，A，B/100.0，1.56，2.37，4.5，0.0/

这个DATA语句是将实型数100.0、1.56、2.37、4.5和0.0分别赋给对应位置的实型变量X、Y、Z、A和B。这个数据语句也可以写成：

```
DATA X, Y, Z/100.0, 1.56, 2.37/
DATA A, B/4.5, 0.0/        或
DATA X, Y, Z/100.0, 1.56, 2.37/, A, B/4.5,
0.0/
```

如果要对字符变量赋初值，使用DATA语句也很方便。如：

```
CHARACTER CH*5, CH1, CH2*3
DATA CH, CH1, CH2/'ABCDE', '␣',
'123'/
```

在常数表中，如果连续出现 n 个相同的常数，则可写成：

n＊〈常数值〉

其中：n为重复因子，它指出相同常数值的个数，重复次数与常数值之间用星号"＊"间隔。如：6＊1.0,表示1.0重复使用6次。

```
DATA A, B, C, D/4*1.5/
```

这表示将1.5分别送到变量A、B、C、D中。

在程序单位中要用DATA语句对变量赋初值，DATA语句必须出现在所有说明语句之后第一个语句函数语句或可执行语句之前。如：

```
DATA A, B, C, X/3.0, 5.0, -2.0, 2.0/
FX=A*X*X+B*X+C
WRITE(*, '(1X, ''F(X)='', F6.1)')FX
END
```

注意：形参、公用块中的变量和函数名不能用DATA语句赋初值。

2.3 简单的输入输出

2.3.1 输入输出语句

在程序中有时要输入数据，有时又要输出数据，这就要用到输入输出语句。

格式：READ(u，f)〈输入表〉

WRITE(u，f)〈输出表〉

其中：READ、WRITE分别是输入输出语句的语句定义符。

u为部件号，f称为数据的输入或输出格式标识符。

1.部件号

部件号(又称为设备号)指所使用的外部设备的代号，它不同于UCSD操作系统下的卷号(如：#4等)。

在计算机系统上都配备有基本的输入输出设备，如：宽行打印机、软磁盘、键盘和显示(监视)器等，可用整数为它们编号，以后要使用这些设备时，给出设备编号就行了。

FORTRAN语言系统规定：

在输入语句中，用O或*表示键盘。但在输出语句中却代表显示器(屏幕)。

部件号u可以是一个整型变量、整型常数或整型表达式。如果是整型变量或整型表达式，那么变量必须事先被赋值。

2.格式标识符

格式标识符为输入输出语句提供数据传输的格式。格式标识符可以是一个标号，它由格式语句FORMAT定义；可

以是格式编辑字符串（字符常数）；也可以是赋有格式编辑字符串的字符变量以及已赋FORMAT语句标号的整型变量。

3.输入输出表

输入输出表由一个或多个元素组成，相互间用逗号“，”隔开。

输入输出表中所有元素的序列可以被输入输出格式作为一个数据或分成多个数据。也就是说，执行一个输入或输出操作意味着输入或输出一个数据或多个数据，并且是从一个新的数据开始的。

值得注意的是：输入表中的元素只允许是变量名。我们来看下面的输入输出语句：

a. READ(*，'3F5.2)'X，Y，Z

b. CH1='(2F5.1，I4)'

   READ(*，CH1)XYZ，ABC，I123

c. WRITE(*，'(3F5.2)')X，Y，Z

d. WRITE(*，IFR)' XYZ= '，XYZ，' ␣␣ABC='，ABC

上述输入输出语句中的变量均遵循隐含(I-N)规则。

执行语句a，计算机要求从键盘上用直接格式输入一个新数据开始的三个实型数据，并分别按照输入的顺序赋给X、Y和Z。

执行b后，将由格式变量决定从键盘上输入两个实型数分别赋给实型变量XYZ和ABC，最后一个为整型数，赋给整型变量I123。

执行语句c，将实型变量X、Y、Z的值从相应的内存储器中按格式输出到屏幕上。

最后一个与语句前面的输出不同，即输出表中出现了字符型常数。当该语句被执行时，则按格式变量提供的格式首先将字符常数“XYZ=”显示在屏幕上，接着再将实型变量XYZ的值从内存取出并按格式变量IFR提供的格式显示。同理，可输出字符常数“␣␣ABC=”和变量ABC的值。

例2 对于f（x）=ax^2+bx+c，给出若干组a、b、c及x的值，分别求每组值对应的f（x）值。

源程序如下：

```
REAL A,B,C,X
WRITE(*,'(''ENTER A,B,C,X=?'')')
READ(*,'(4F4.1)')A,B,C,X
FX=A*X*X+B*X+C
WRITE(*,'(''F(X)='',F10.4)') FX
END
```

我们可以反复运行这段程序，而每次运行时输入不同的一组值。如：

ENTER A, B, C, X=?

3.0␣5.0␣−2.0␣2.0↲

F（X）=␣␣␣20.0000

2.3.2 格式说明及其应用

格式说明有多种形式，如前面例中所使用的形式就是其中之一。下面将进一步讨论常用的直接格式说明和FORMAT语句提供的格式说明。

1.直接格式说明

格式：′（〈格式说明〉）′

它用单撇号和圆括弧将格式说明括起来，而格式说明又

可以是一个或多个格式码，可将这一格式直接嵌入输入和输出语句中。如：

WRITE (*, '1X, 5HF (X) =, F10.2) ') FX

按照这种格式说明输出结果如下：

␣F (X) =␣␣␣␣␣␣20.00

从上面的输出语句中已经看到格式码的作用和简单用法。那么，怎样编写所需要的格式说明，选用那些格式码以及格式码与输入输出表中变量的关系如何呢？下面将分别予以介绍。

2. 常用的格式码

格式码是格式说明的基本组成部分（或称为格式元素），按其功能分成可重复的格式码和不可重复的格式码。利用格式码可自由组合输入输出格式。常用的格式码主要有I、F、E、X、H、' 和/。

（1）整型格式码I

整型格式码I用来规定整型数据的输入和输出。它的一般形式为：

rIw

其中：r、w均为正整数。w用来指定整数的位数，称为数据的输入输出宽度，又称为单场宽（或字段宽度）。

r是重复因子。当多个整数选用同样的I格式码进行输入输出时，可以不必重复书写多个相同的格式码，而采用r重复因子，指出重复使用的次数，如：

READ (*, ' (I3, I3, I3) ' I, J, K也可写成：

READ (*, ' (3I3) ') I, J, K

这个输入语句的含义是从键盘上接收三个整型变量I、J、K的值，每个值占三位，I3表示每个整数占3位，3I3表

示这种格式重复执行三次。

若输入数据为：

123456789↵

则　I=123，J=456，K=789

使用格式码输入数据时应注意：

输入数据中有效的空格等于零；

正或负号放在数值的前面，正号可以省略；在输入正负号时，正号或负号均要计入格式码所指的宽度内；

输入的数据应按I格式码“向右看齐”，不足w时，应在数字左边补空格或零；

若输入表中元素的个数多于格式码的个数，则一个输入数据的长度等于所有格式码场宽的总和。

对于输出语句中的格式说明与此类似。但对于显示或打印输出语句来说，数据的开头必须提供显示或打印的控制符，否则系统将数据的第一个字符当作控制符使用，从而造成显示或打印输出数据的第一个字符被“吃”掉。在UCSD PASCAL操作系统下的FORTRAN II.I则不存在此问题。

用I格式码输出时，FORTRAN语言规定：

输出数据中元素之间没有空格符，因此，编排格式说明时，可将字段宽度加大一些。

数字“向右看齐”输出。输出数据的位数少于字段宽度w时，则左边补空。

输出数据中的数为正时，则正号被省略；如果为负数，则在数字前打印出负号并计入字段的宽度中。

如果输出字符宽度大于给定的字段宽度w，则打印输出w个星号“*”。

（2）实型格式符F

实型格式符用来指定实型数的输入输出。其一般形式是：

rFw.d

其中：r、w的含义同I格式码。

d表示输入输出实数的小数部分的位数。

在F格式符中，字段宽度w称为第一字段宽度（总宽度），d称为第二字段宽度。又称F格式符为双字段宽的格式码。

用F格式符传输实型数据的形式是：

$$\pm\ \overbrace{**\cdots**.\underbrace{**\cdots**}_{d}}^{w}$$

输入四个实型变量A，B，C，X的值，语句可写成：

READ(*，'(F4.1，F3.1，F4.1，F3.1)')A，B，C，X或

READ(*，'(4F4.1)')A，B，C，X

若前者的输入数据为：

␣3.05.0−2.02.0↵

则：A＝3.0，B＝5.0，C＝−2.0，X＝2.0

后一个语句是将前一个语句的格式符统一为“4.1”的字段宽度，于是在F4.1格式符前带重复因子，即重复执行F4.1格式符四次。F4.1意思是每个实数占四位，其中有一位小数。

实际使用中如何确定F格式符的w和d呢？可用下列公式估计：

当实型数小于0时，w≥n＋d＋2

当实型数大于或等于0时，w≥n+d+1

如果在宽度许可的情况下，可统一用公式

$$w \geq n+d+2$$

来确定第一字段宽。

其中，n为整数部分的位数，d为小数部分的位数，数字2代表小数点和符号的位数。

（3）指数型格式符E

除实型格式符F以外，还有一种格式，其输入和输出的数据为指数形式：

$$\overbrace{\pm 0.\underbrace{**\cdots**}_{d}E\pm **}^{w}$$

格式符的形式是：

rEw.d

其中r、w、d的意义同格式符F。

用E格式符，输入输出数据可以适应不同大小的数值。用于输出时，它可以有足够的有效位数字，输出精度比F格式要高得多。只要传输的数据在实数范围内，均可得到正确的值，而不存在超长度的问题。

下面是E和F格式的比较：

变量值	格式符	输　出
234567890123.75	E14.8	␣.23456789E+12
	F14.8	**************
−0.0003756	E13.6	␣−.375600E−03
	F14.7	␣␣␣␣␣␣−.0003756

如果输入语句中使用E格式符，则对于大数据的输入也

是方便的。

输出数据宽度W的确定：

当数据值≥0时，W≥d+6

当数据值<0时，W≥d+7

一般都选择后者较为安全。

（4）空格格式符X

空格格式符的形式是：

nX

其中：n为正整数。

在输入语句READ中，nX格式符使数据位置向前移动n个字符。因此从数据的当前位置开始的n个字符被跳过。

在输出语句WRITE中，nX格式符从当前位置开始输出n个空格，使输出的结果前后之间有一定空格易于阅读。

nX格式符是不可重复的格式符。因此，不能在nX前再加重复因子。输入输出表中也没有与之对应的项，请看下面的示例：

```
WRITE(*,'(1X2F6.2, I4)')A, B, J
```

如果A=345.23，B=456.78，J=9123，则输出结果为：

345.23456.789123　　　（显然，数据间无空格）

若将输出格式改写为：

```
WRITE(*,'(1X, 2F8.2, 3X, I4)')A,B, J
```

则输出结果为：

```
␣␣345.23 ␣␣456.78 ␣␣␣ 9123(数据间有空格)
  F8.2     F8.2    3X   I4
```

（5）文字格式符H

文字格式符可用于输出格式符后面出现的全部字符。其一般形式为：

$$nH\ \underbrace{h_1h_2\cdots hn}_{n个字符}$$

其中：n为正整数，它指明H后字符的实际个数。

hi（i=1，2…，n）代表一个字符。如：

WRITE（＊，'（1X，2HA=，F6.2，3X，2HB=，F6.2，5H␣␣␣J=，I4）'）A，B，J

如果变量的值如前例，则输出结果为：

$$\underbrace{A\ =}_{2H}\ \underbrace{345.23}_{F6.2}\ \underbrace{␣␣␣}_{3X}\ \underbrace{B=}_{2H}\ \underbrace{456.78}_{F6.2}\ \underbrace{␣␣␣J=}_{5H}\ \underbrace{9123}_{I4}$$

（6）撇号格式符“'”

撇号格式符具有字符常数的形式，它用于直接输出字符串。如：

WRITE（＊，'（1X，''CHINESE␣EDUATION␣COMPUTER␣␣CEC—I''）'）

输出结果是：

CHINESE␣EDUCATION␣COMPUTER␣␣CEC—I

（7）斜杠格式符“/”

斜杠格式符指明当前的数据传输结束（末端）。对于输入数据，斜杠格式符的作用是跳过当前数据的剩余部分，定位于下一个输入数据的开始。对于输出数据，斜杠格式符结束当前数据并定位于下个数据的开始。当用于打印格式说明时，连续n个斜杠表示打印n－1个空行。如：

```
WRITE(*,'(///, 20X,''ZHONG␣␣␣HUA
␣␣␣XUE
*␣␣␣XI␣␣␣JI'',////,30X,''CEC-I'',
///)')
```

请读者自己考虑这个打印语句的输出结果是什么格式。

3.格式语句

格式语句是向输入输出语句提供格式说明的一种非执行语句，也是一种说明语句，它和其它说明语句不同，它可以出现在任何可执行语句之间。但它必须出现在使用该语句的程序单位中。

格式语句的形式是：

〈标号〉FORMAT(〈格式说明〉)

其中〈格式说明〉由格式符序列组成。FORMAT语句的前面（标号区）必须有标号，标号为输入输出语句引用格式提供信息。

在输入输出语句中，输入输出表中变量的类型要与格式说明中格式符的类型一致。同一个格式语句可被多个输入输出语句所引用，即共享同一格式。

在程序设计过程中，往往将所有的格式语句集中放于本程序单位的尾部，以增加程序的可读性。

例3 编写求解一元二次方程$ax^2+bx+c=0$的两个实根的程序。

求根公式为：

$$x_{1,2}=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$$，当$b^2-4ac\geq 0$时有两个实根。

其中a，b，c分别为方程的系数。下面是求根的一个源程序。

```
C THE ROOTS OF THE QUADRATIC EQUATION.
      WRITE(*,100)
      READ(*,200) A,B,C
      D=SQRT(B*B-4.0*A*C)
      X1=(-B+D)/(2.0*A)
      X2=(-B-D)/(2.0*A)
      WRITE(*,300) A,B,C
      WRITE(*,400) X1,X2
100   FORMAT(1X,'ENTER A,B,C=?')
200   FORMAT(3F5.1)
300   FORMAT(1X,'A=',10X,'B=',10X,'C=',
     1/1X,3(F5.1,7X),//)
400   FORMAT(1X,'X1=',17X,'X2=',
     1/1X,2(F14.5,6X))
      END
```

选定一组系数后，运行结果如下：

```
ENTER A, B, C=?
␣␣2.0␣10.0␣␣5.0↲
A=            B=          C=
␣␣2.0␣␣␣10.0␣␣␣␣5.0


X1=                    X2=
      -.56351                -4.43649
```

习题二

1.下列赋值语句哪些是正确的？哪些不正确？为什么？

X=3.14159/180, A-B=X+Y+0.5

f(X)=X*X+Y*Y, X123=(A*A+B*B)**(-2)

2.把下列各式写成赋值语句

$A=\sqrt{X+Y}$, $CH=|X_1+X_2|<\varepsilon$, $L=X\geqslant Y$

3.下列输入输出语句正确吗？为什么？

```
READ*, 'ABCD', 'F(X)='
WRITE(10, '(F5.2))' X, Y, Z
```

4.编写程序，

求：$$\frac{\sin(|X|+|Y|)}{\sqrt{\cos(|X+Y|)}}$$

其中：$X=22.5°$，$Y=13.4°$

第三章 FORTRAN程序的运行

现在我们已经能够书写简单的FORTRAN程序了。为使读者能够尽快的在CEC-I机上编写和运行FORTRAN程序，我们暂时中断对语言的介绍，转而介绍如何输入、修改、编译、连接和运行程序。

中华学习机CEC-I是与APPLE-Ⅱ兼容的计算机系统。我们成功地将APPLE-Ⅱ在UCSD PASCAL操作系统下的FORTRAN77引入到CEC-I中，实现了该系统下的全部功能。下面介绍UCSD PASCAL操作系统下FORTRAN语言在CEC-I机上的运行处理方法。

3.1 UCSD PASCAL下的FORTRAN配置

UCSD PASCAL操作系统下的FORTRAN77是一个功能较强的高级程序设计语言，与FORTRAN77有关的全部文件分别存放在两张单面软盘上（软盘名分别称为FORT1：和FORT2：）。

FORT1：盘上有下列文件：

FORTLIB·CODE	FORTRAN77库编辑程序。
SYSTEM·APPLE	用6502机器语言编写的P代码解释程序的数据文件。
SYSTEM·PASCAL	UCSD PASCAL操作系统

	主程序。
SYSTEM·MISCINFO	使用的终端信息，告诉系统有哪些终端设备。
SYSTEM·CHARSET	图形字符集。
SYSTEM·FILER	文件管理程序，用于存入、删除和传送文件等。
SYSTEM · EDITOR	文本编辑程序。
SYSTEM·LINKER	FORTRAN连接程序。

FORT2：盘上存放有下列文件：

SYSTEM·LIBRARY	FORTRAN77程序库。
SYSTEM · PASCAL	同FORT1盘。
SYSTEM·MISCINFO	同FORT1盘。
SYSTEM·CHARSET	同FORT1盘。
SYSTEM·COMPILER	FORTRAN77语言编译程序，可将FORTRAN源文件转换成P 代码文件。
SYSTEM · APPLE	同FORT1盘。

其中，FORT2：是UCSD PASCAL操作系统的主文件软盘，我们称为系统引导盘。

如果你的计算机系统具有多个软盘驱动器，则可将FORT2：作为引导盘放在#4：驱动器上，而将FORT1：盘放在#5：驱动器上。这样，当启动UCSD PASCAL后，系统会自动到#4：和#5：上去调用所需要的文件，从而可以实现对FORTRAN77源程序文件的输入、编辑、编译、连接和运行。

3.2 CEC-1上的FORTRAN配置

3.2.1 CEC-I需要哪些系统文件

从上节中已经看到：UCSD PASCAL操作系统下FORTRAN77的全部文件分别保存在两张盘上，这对于多个驱动器来说使用是方便的。但要应用于仅有一个驱动器的中华学习机，就有困难了。即使可以用交换盘片的方法进行工作，但最后连接生成代码文件时仍然出错：

CODE WRITE ERR

TYPE 〈SP〉(CONTINUE)，〈ESC〉(TERMINATE)

在分析了FORTRAN系统配置后，就可以根据用户的需要来重新构造系统软盘。在这之前，用户要知道以下两点：

1.UCSD PASCAL操作系统的基本文件

SYSTEM·APPLE

SYSTEM·PASCAL

SYSTEM·MISCINFO

SYSTEM·CHARSET

通过它们可引导进入UCSD PASCAL操作系统。

2.FORTRAN的工作文件

在UCSD PASCAL操作系统下进行FORTAN77源文件的建立、编辑、编译、连接、运行以及对磁盘文件的管理，必须有下列文件：

（1）能够建立或修改FORTRAN77源程序的文本编辑

程序SYSTEM·EDITOR。

（2）能够编译FORTRAN77源程序文件的FORTRAN77编译程序SYSTEM·COMPILER，它将FORTRAN77源程序文本文件翻译成P代码文件。

（3）连接P代码文件和FORTRAN77库文件中的子程序，需要连接程序 SYSTEM·LINKER 和 FORTRAN77 库文件 SYSTEM·LIBRARY。它将指定的P代码文件改造为一个可供执行的代码文件。

（4）能够进行磁盘文件管理的SYSTEM·FILER文件。

以上提到的四类系统文件，其总容量已超过一张软盘的最大容量143K。因此，只能放在两张软盘中，分别作为编译和连接盘。除将与编译或连接FORTRAN程序有关的文件存放于各自的磁盘上之外，还必须具备UCSD PASCAL操作系统的基本文件。

3.2.2 如何构造CEC-I的系统文件盘

明确上述两点以后，就能够构造适用于CEC-I的FORTRAN软磁盘了。其操作过程如下：

（1）将FORT2：软盘上的全部文件分别复制到两张空白软盘上。由于SYSTEM·LIBRARY不能进行单独复制，因此，必须用拷贝软件(如COPY V8.0等的整盘复制功能)将FORT2：整盘复制到两张空白盘中。

（2）将复制的这两张盘的盘名分别改为 CECF1：和 CECF2：。

（3）将CECF1：作为连接盘，因此删去 SYSTEM·COMPILER程序。然后将FORT1：软盘上的 SYSTEM·FIL

ER和SYSTEM·LINKER两个程序拷贝到CECF1：盘中。

（4）将CECF2：作为编译盘，因此可删去SYSTEM·LIBRARY程序。然后将CECF1：软盘上的SYSTEM·FILER和SYSTEM·EDITOR两个程序拷贝到CECF2：盘中。

至此，已构造好两张FORTRAN系统软盘CECF1：和CECF2：，各自的文件分别为：

CECF1：	CECF2：
SYSTEM·LIBRARY	SYSTEM·EDITOR
SYSTEM·PASCAL	SYSTEM·PASCAL
SYSTEM·FILER	SYSTEM·FILER
SYSTEM·MISCINFO	SYSTER·MISCINFO
SYSTEM·CHARSET	SYSTEM·CHARSET
SYSTEM·LINKER	SYSTEM·COMPILER
SYSTEM·APPLE	SYSTEM·APPLE

系统建立以后，可作FORTRAN77源程序文件的编辑、编译、连接等工作。

3.3 FORTRAN77程序的上机过程

UCSD PASCAL操作系统下的FORTRAN语言是一个编译型的软件。如何在中华学习机上运行FORTRAN程序呢？下面将分别介绍它的每一步操作。

3.3.1 启动UCSD操作系统

启动UCSD操作系统有两种方式。

1.加电启动自动进入UCSD命令级

当你按正确的方法将各种设备连接好以后，首先打开电视机（或显示器）的电源开关，然后，将含有UCSD操作系统文件的软盘CECF2：插入磁盘驱动器中，关好驱动器小门，再打开主机电源开关。此时可听到主机喇叭发出的“嘟”声，同时，键盘右上方的指示灯亮。紧接着系统将启动软盘驱动器，并从CECF2：盘中装入操作系统。

当UCSD操作系统被装入内存并运行后，这时将在电视机（或显示器）的屏幕顶部显示操作系统命令级的提示信息：

COMMAND：E（DIT，R（UN，F（ILE，C（OMP，L（INK，X（ECUTE，A（SSEM，D（EBUG？

这意味着UCSD操作系统已经运行，并且等待用户敲入所需命令的第一个字母。如：敲字母“E”，则执行文本编辑程序。

2.进入基本系统后的启动

当你按正确的操作启动基本系统（软盘驱动器小门开着），主机自动进入西文BASIC状态，屏幕显示如下：

```
ZHONG  HUA  XUE  XI  JI
      VERSION  1.1
]■
```

图中的“]”为西文BASIC语言的提示符，紧接的小方块被称为光标。这意味着系统可接收用户输入的BASIC命令了。

由于我们需要进一步引导UCSD操作系统，因此，可将含有UCSD操作系统文件的软盘CECF2：插入驱动器，关

好小门并键入命令：

]PR#6↵

当你敲回车键后，BASIC系统解释该命令并启动磁盘机，从软盘上读入UCSD操作系统程序并运行。此时，屏幕顶部显示出命令级的提示信息（同1）。

通过上述两种方式启动UCSD操作系统后，系统等待用户进行各种操作。对于FORTRAN语言程序的运行，还必须作下列几步操作：

（1）执行文本编辑程序SYSTEM·EDITOR，建立和修改FORTRAN源程序文本文件·TEXT，并将该文件存放在CECF2：盘上。

（2）执行CECF2：盘上的FORTRAN编译程序SYSTEM·COMPILER，对FORTRAN源文件进行编译。如果编译时无错误，则产生P代码文件·CODE；如果有错误，则可执行文本编辑程序，对该FORTRAN源文件进行修改并再次执行本步操作。

（3）执行文件管理程序SYSTEM·FILER中的T操作，将第2步产生的P代码文件传送到连接盘CECF1：上。

（4）执行连接程序SYSTEM·LINKER，将P代码程序连接（改造）成可执行的代码文件·CODE。

（5）运行第4步产生的可执行程序。

3.3.2 源程序的输入和修改

在CECF2：盘上的SYSTEM·EDITOR程序是一个编辑文本文件的系统实用程序。在UCSD PASCAL操作系统的COMMAND命令级，可键入字母E执行文本编辑程序，在SYSTEM·EDITOR程序的控制下，即可实现FORTRAN

源程序的输入。

1.启动文本编辑程序

在UCSD PASCAL操作系统的命令级下键入字母E，则进入编辑程序并在屏幕顶部显示编辑提示信息如下：

>EDIT：A(DJST C(PY D(LETE F(IND I(NS J(MP NO WORKFILE IS PRESENT FILE?

提示信息的第一行为编辑程序的功能选择，这里可按如下方式选择回答：

（1）如果要编辑磁盘上已存在的文本文件，则可直接输入文件名，然后敲回车键“↲”将指定的文件读入内存（此时的文件名可以不带扩展名·TEXT，系统自动找到带扩展名·TEXT的文本文件），并显示文本文件开始部分的内容。

（2）如果是建立新文件，则先敲回车键↲，然后敲字母I键，系统进入允许插入状态，出现插入状态下的提示，这时就可以输入源程序的语句字符了。

（3）若敲ESC+↲（即按ESC和↲，下同）键后，则返回到COMMAND命令级状态。

2.对源程序的处理

FORTRAN源程序的处理，可分两种情况：

（1）如果是新建一个FORTRAN源程序，则首先应进入插入状态，在此状态下按照FORTRAN源程序的格式输入所有的语句字符，输入完后敲CTRL+C以结束插入状态。若需要进行修改则进入编辑状态。

（2）如果对已存在的源程序进行编辑，则分别按不同的选择进行。

若要删除某些字符，则应键入字母D进入删除方式，然

后敲光标控制键“◁”和“▷”，可分别删除当前光标左边或右边（包括光标下）的字符。当未结束删除而光标又回退时，则恢复被删除的字符。删除结束后敲CTRL＋C。敲CTRL＋C后，删除的字符就不能再恢复了。

若要插入某些字符，可将光标移到要插入字符的位置，然后敲字母I，则进入插入状态。这时可将添加的语句、字符等插入到此处。同样，敲CTRL＋C可以结束插入状态。

其它功能选择可根据需要而定。

在进行源程序的编辑处理时，需要移动光标以及其它一些控制键配合进行。常用的光标移动控制键有：

CTRL＋C	中断处理或退出当前编辑状态。
▷	向右移动光标。
◁	向左移动光标。
CTRL＋O	光标上移一行。若要移动多行，可先键入行数，再敲CTRL＋O。如：3CTRL＋O，则当前的光标上移3行。
CTRL＋L	光标下移一行。若要下移多行，则先键入行数后再敲CTRL＋L。如：5CTRL＋L，则光标下移5行。
CTRL＋I	光标向前跳过8个字符。
CTRL＋A	控制屏幕的左移或右移。

3.退出编辑程序

退出编辑程序时，当前应处于编辑功能选择状态。若不是则敲CTRL＋C，使之退到该状态。此时可选择退出命令Q（即QUIT）。当敲字母键Q后，则显示如下提示：

＞QUIT

U（PDATE THE WORKFILE AND LEAVE

E(XIT WITHOUT UPDATING
R(ETURN TO THE EDITOR WITHOUT UPDATING
W(RITE TO A FILE NAME AND RETURN
S(AVE WITH SAME NAME AND RETURN

这里有五种退出编辑程序的方法：

（1）选择U命令时，则将编辑缓冲区的信息以ASCII码形式存磁盘上，其文件名为SYSTEM·WRK·TEXT。然后返回到操作系统的COMMAND命令级状态。

（2）选择E命令时，则返回到操作系统的命令级，编辑缓冲区的信息不存入磁盘，即不修改原先的文本文件。

（3）选择R命令时，则退回到编辑状态下继续编辑，编辑缓冲区先前的信息不被破坏。

（4）选择W命令时，则显示下列提示：

>QUIT:
NAME OF OUTPUT FILE(〈CR〉TO RETURN)→

输入合法的文件名后敲“↵”，则将编辑缓冲区的信息存入指定的文件中，然后退出编辑程序并回到COMMAND命令级状态。

若直接敲入↵，则重新回到编辑状态。

（5）选择 S 命令时，则从退出状态转向保存文件的操作。可分两种情况进行：

a.编辑的是新文件，将自动把编辑缓冲区的信息存入盘上的SYSTEM·WRK·TEXT文件中。

b.如果编辑的是由文件管理SYSTEM·FILER获得的用户文件名，则执行S命令时，系统显示下列提示信息：

PURGE OLD 文件名 BEFORE SAVE?

如果回答Y，则将编辑缓冲区的内容存入原有文本，但磁盘文件名不变。如果回答N则将编辑缓冲区的内容存入SYSTEM·WRK·TEXT文件。

以上五种退出编辑程序的方式，在写磁盘文件遇到磁盘存储空间不够时，将显示出如下提示：

ERROR：WRITING OUT THE FILE PLEASE PRESS

〈SPACEBAR〉 TO CONTINUE

此时只能敲空格键返回编辑状态，再选择其它退出方式。

例1　输入第二章例3求解一元二次方程实根的程序，并将源程序用ROOT名字保存在磁盘上。

FORTRAN程序的编辑过程如下：

在UCSD命令级敲字母E命令。

E

>EDIT：A(DJST C(PY D(LETE F(IND I(NS J(MP NO WORKFILE IS PRESENT FILE?

↵ I（敲回车键后再输入I命令）

>INSERT：TEXT[〈BS〉A CHAR, 〈DEL〉A LINE]

```
C THE ROOTS OF THE QUADRATIC EQUATION.
        WRITE(*,100)
        READ(*,200) A,B,C
        D=SQRT(B*B-4.0*A*C)
        X1=(-B+D)/(2.0*A)
        X2=(-B-D)/(2.0*A)
        WRITE(*,300) A,B,C
        WRITE(*,400) X1,X2
 100    FORMAT(1X,'ENTER A,B,C=?')
 200    FORMAT(3F5.1)
 300    FORMAT(1X,'A=',10X,'B=',10X,'C=',
       1/1X,3(F5.1,7X),//)
```

```
400    FORMAT(1X,'X1=',17X,'X2=',
      1/1X,2(F14.5,6X))
       END
```

敲CTRL＋C后再敲Q，系统显示提示信息：

>QUIT：

U(PDATE THE WORKFILE AND LEAVE

E(XIT WITHOUT UPDATING

R(ETURN TO THE EDITOR WITHOUT UPDATI NG

W(RITE TO A FILE NAME AND RETURN

S(AVE WITH SAME NAME AND RETURN

此时键入W后出现提示：

NAME OF OUTPUT FILE(〈CR〉 TO RETURN)→

（提示信息的意思是要求输入文件名并带扩展名·TEXT。若无扩展名，系统在文件名后自动加上扩展名·TEXT）

输入文本文件名：ROOT，则将编辑缓冲区的程序存到CECF2：磁盘上，文件名为ROOT·TEXT。

3.3.3 编译源程序

编译FORTRAN77源程序，生成P代码文件以作为SYSTEM·LINKER文件的输入文件。其过程如下：

C（用户键入编译命令）

COMPILING…　　　　（系统提示信息）

COMPILE WHAT TEXT？ROOT↲　　（输入要编译的文件名）

TO WHAT CODEFILE？ROOT↲　　（P代码文件名。若用“$”代替，则P代

码文件名与编译文件名一致）。

LISTING FILE? ↙　　（输入列表文件名，若不需要则直接敲↙。也可输入CONSOLE：或PRINTER：，前者在屏幕上显示列表文件而后者则用打印机输出列表文件）。

FORTRAN77 －（C）1980 SILICON VALLEY SOFTWARE, INC.16-MAY-80 FORTRAN COMPILER II.I〔1.1〕

〈　　0〉…

NONAME〔　　　WORDS〕

〈　　2〉……

14 LINES. O ERRORS.

SMALLEST AVAILABLE SPACE=　　　WORDS.

到这时，编译程序将源程序ROOT·TEXT翻译成P代码文件并存储在ROOT·CODE文件中。这个代码文件还不能执行，必须连接之后才能运行。

3.3.4 连接P代码文件

在连接由编译程序输出的P代码文件之前，必须通过文件管理程序SYSTEM·FILER的传送命令T将P代码文件从编译盘（CECF2：）上传送到连接盘（CECF1：）上，将CECF1：保留在驱动器上。然后在UCSD命令级键入字母“L”，则运行连接程序。其过程如下：

1.从编译盘上将P代码文件传送到连接盘上

F　　（用户在UCSD命令级敲入文件管理命令）

此时在屏幕顶部提示行出现文件管理子命令：

FILER：G，S，N，L，R，C，T，Q〔1.1〕

TRANSFER？ROOT·CODE，#4：ROOT·CODE↲

（后一个ROOT·CODE可用字符“$”代替，表示与前者相同）

INSERT DESTINATION DISK
TYPE 〈SPACE〉 TO CONTINUE■
CECF2：ROOT·CODE
-->CECF1：ROOT·CODE

（将CECF2：盘从驱动器中取出并插入CECF1：盘后敲入空格键，则将CECF2：ROOT·CODE转到CECF1：盘中）

Q　　（用户在文件管理控制下输入文件管理子命令Q退出并返回到UCSD命令级）

2.连接P代码文件

到此已作好了连接P代码文件的准备工作。下面是连接的过程：

L　　（用户在系统命令级输入连接命令L，系统读入连接程序SYSTEM·LINKER到内存并执行）

LINKING…　　（连接程序输出信息）

APPLE PASCAL LINKER〔1.1〕

HOST FILE？ROOT↲　　（要求输入P代码主文件名）

OPENING ROOT·CODE

LIB FILE？* ↲　　（回答“*”则代表FORTRAN库

文件）

OPENING * SYSTEM·LIBRARY

LIB FILE? ↲　　（可输入其它库文件名，最多8个，没有则直接敲↲）

MAP FILE?　　（输入映象文件名，不需要则敲↲）

READING MAINSEGX

READING NONAME　　（读入用户P代码文件中的程序）

READING RTUNIT　　（读入库中运行子程序）

OUTPUT FILE? ROOT↲　　（输入可执行文件名）

LINKING NONAM #7

LINKING RTUNIT #8

LINKING MAINSEGX #1

这就将P代码文件连接成了一个可供执行的代码文件ROOT·CODE，并将其存储在连接盘CECF1：上。

在连接程序的过程中读入了3个程序块：

#1号MAINSEGX：这个文件中含有用于管理所有其它段、定义有名公共块、初始化运行时的系统代码。

#7号NONAME为用户的P代码文件中的程序名。

#8号RTUNIT为FORTRAN程序库中的运行子程序。

3.3.5　执行一个用户程序

一个由FORTRAN77源程序最后生成的代码文件，必须由 UCSD PASCAL 操作系统命令X将连接好的代码文件装入内存运行。当敲入命令X后，则屏幕显示提示信息：

EXECUTE WHAT FILE? ROOT↲

当输入文件名ROOT·CODE并敲↲后就开始执行，并

显示程序输出的结果：

```
ENTER A, B, C=?
␣␣2.0␣10.0␣␣5.0↲
A=          B=            C=
␣␣2.0␣␣␣␣10.0␣␣␣␣␣5.0
X1=                        X2=
    -.56351                    -4.43649
```

习题三

1. 简略说明在UCSD操作系统下FORTRAN程序的执行过程。

2. 编程序，

$x^2-2x+1=0$

并上机计算输出x的值。

第四章　控制语句及分支程序

从前面的示例中可以看到，一个FORTRAN程序总是从主程序的第一个可执行语句开始按语句出现的先后次序往下执行的。但是，有时需要改变程序执行的顺序，例如：要跳过某一部分语句，或重复执行某些语句，或根据条件选择执行某些语句。本章将介绍这些控制语句以及分支程序的编写方法。

4.1　转移语句

转移控制语句，主要用于改变程序的执行顺序，是实现分支程序设计不可缺少的一类控制语句。转移控制语句分为：无条件转移、计算转移和赋标号转移。

1.无条件转移语句

格式：GOTO〈标号〉

作用：无条件转移到指定标号所标识的语句去执行。其中标号是此语句所在的程序单位内的一个可执行语句的标号。

无条件转移语句的使用比较简单。如果需要转向某个语句去执行，只需在该语句前添上语句标号，再在该程序单位内适当的地方写上无条件转移语句GOTO〈标号〉即可。

2.计算转移语句

无条件转移语句只能转移唯一的一个标号而毫无选择的

余地。为能满足多分支的处理，再介绍计算转移语句。

格式：GOTO（L_1，L_2，…，Ln）〔,〕I

作用：根据I的值，可以转到相应的Li去执行。

这里I可以是一个已赋值的整型变量或一个整型算术表达式（式中变量必须是已赋值的）。Li（i=1，2，…，n）为可执行语句的标号，它们与计算GOTO语句在同一程序单位内。同一个标号可在标号表中出现多次，I的值指出要转移的语句标号在标号表中的位置序号。

注意：如果I<1或I>n，则顺序往下执行。

例1　求f(x)的值：

$$f(x)=\begin{cases} x^2 & \text{当}0\leqslant x<1\text{时} \\ x^2-1 & \text{当}1\leqslant x<2\text{时} \\ x^2-2x+1 & \text{当}2\leqslant x<3\text{时} \\ x^2+4x-17 & \text{当}3\leqslant x<4\text{时} \end{cases}$$

函数f(x)有四个计算分支，每个分支的自变量x的变化区间都是1。我们用I来控制使之能够转移到相应的分支，每个分支给出一个语句标号。FORTRAN程序如下：

```
      INTEGER I
10    WRITE(*,'(1X,''INPUT ARGUMENT X=?''\')')
      READ(*,'(F4.2)') X
      I=X+1
      GOTO (20,30,40,50) I
      WRITE(*,'(1X,''X<0 OR X>=4'')')
      GOTO 10
20    F=X*X
      GOTO 100
30    F=X*X-1.0
      GOTO 100
40    F=X*X-2*X+1.0
      GOTO 100
```

```
50      F=X*X+4*X-17.0
100     WRITE(*,'(1X,''X= ',F4.2,
     1  '    F=  ',F16.7)') X,F
        END
```

如果输入数据分别为0.5、1.3、2.6、35和6时，则程序运行的结果如下：

INPUT ARGUMENT X=?

0.5↵

x=␣.50␣␣␣F=␣␣␣␣␣␣.250

INPUT ARGUMENT x=?

1.3↵

x=1.30   F=   .690（以下略）

当输入变量x的值不符合要求时，则再次要求输入自变量x的值，否则程序将处于等待状态。

3.赋标号语句

赋标号语句是将可执行语句或格式（FORMAT）语句的标号值赋给一个整型变量。

格式：ASSIGN〈标号〉TO〈变量〉

作用：将语句标号赋给一个整型变量。

其中：标号是一个格式语句的标号或可执行语句的标号。变量必须是一个整型变量。

执行赋标号语句ASSIGN的实际效果是将可执行语句标号值或格式语句标号值赋给整型变量，被赋标号值的整型变量可以用于赋值GOTO语句中，也可以作为格式标识符出现在输入输出的语句中。被赋的语句标号必须是与ASSIGN语句在同一程序单位中的可执行语句的标号或格式语

句的标号。如：

```
      PROGRAM ASSIG1
      ASSIGN 100 TO I
            ⋮
      ASSIGN 1000 TO K
            ⋮
100   FORMAT(1x, …)
            ⋮
1000  J=J+1
      WRITE(*, I)…
      END
```

这就是把标号100，1000分别赋给了整型变量I和K，而WRITE语句中就用到了I。

注意：赋标号语句ASSIGN是将可执行语句或格式（FORMAT）语句的标号值赋给整型变量的唯一方法。

4.赋值转移语句

（1）赋值转移语句的简单形式为：

GOTO I

其中，I为已赋语句标号值的整型变量。

从语句形式上看，赋值转移语句的简单形式相当于无条件转移语句。

在执行赋值GOTO语句之前，I必须由本程序单位内的赋标号语句ASSIGN赋值，执行赋值GOTO语句时，则将控制转到以I的值为标号的可执行语句去执行。

（2）赋值转移语句的一般形式

赋值转移语句的一般形式是：

GOTO I, (L_1, L_2, …, L_n)

其中：I为已赋可执行语句标号的整型变量。

Li（i=1，2，…，n）为与赋值GOTO语句在同一程序单位内的可执行语句的语句标号，标号表中可以有相同的语句标号。

在执行赋值GOTO语句时，I的值必须由赋标号语句赋值，其值所标识的可执行语句与赋值GOTO语句在同一程序单位内，且I的值必须在标号表中。若在标号表中找不到I的值，则产生运行错误。赋值GOTO语句不能转入DO、IF、ELSEIF和ELSE块内。

例2　用ASSIGN语句和赋值GOTO语句编写求Y、Z函数值的程序，其中Y、Z为X的函数，它们的关系如下：

$$Y=\begin{cases}(X+1)^2\\(X+2)^3\\(X+4)^4\end{cases}\quad Z=\begin{cases}\dfrac{1}{X+1} & \text{当}-5\leqslant X<0\text{时}\\[2mm]\dfrac{1}{(X+2)^2} & \text{当}0\leqslant X<5\text{时}\\[2mm]\dfrac{1}{(X+4)^3} & \text{当}5\leqslant X<10\text{时}\end{cases}$$

编写这个程序需要进行条件判断，找X所处的区间。由于X的区间是等长的，因此可用ASSIGN和算术GOTO语句实现。程序如下：

```
          ASSIGN 40 TO L
          ASSIGN 400 TO LI
5         READ(*,'(F6.2)') X
          I=(X+5.0)/5.0+1.0
          GOTO (10,20,30) I
          WRITE(*,'(''X<-5 OR X>=10 ERROR !'')')
          GOTO 5
10        ASSIGN 100 TO K
          GOTO L
```

```
20      ASSIGN 200 TO K
        GOTO L
30      ASSIGN 300 TO K
40      GOTO K,(100,200,300)
100     Y=(X+1.0)**2
        Z=1/(X+1.0)
        GOTO LI
200     Y=(X+2.0)**3
        Z=1.0/(X+2.0)**2
        GOTO LI
300     Y=(X+4.0)**4
        Z=1.0/(X+4)**3
400     WRITE(*,60) X,Y,Z
60      FORMAT(1X,2HX=,F4.0,2X,2HY=,
       1F10.4,2X,2HZ=,F10.8)
          END
```

这个程序综合地应用了计算GOTO、赋标号语句ASS-IGN、赋值GOTO语句。

第一、二两个赋标号语句是将赋值GOTO语句和输出语句WRITE的语句标号分别赋给整型变量L和LI，为简单赋值GOTO语句的整型变量赋初值。

计算GOTO语句的三个分支主要为赋值GOTO K，(100，200，300)语句提供计算函数值Y、Z的入口。通过GOTO K，(100，200，300)语句的执行转移到相应的分支函数进行计算，从而得到函数值Y和Z。

最后由输出语句WRITE在屏幕上按指定格式显示出结果。

如果输入的自变量值不在要求的区间内，则显示错误提

示信息：

X<-5 OR X>=10 ERROR!

并处于等待状态。此时，可继续输入X的值。如：输入数据为：

-3.↙（以下为输出结果）

X=␣␣-3.␣␣Y=␣␣␣␣␣4.0000␣␣Z=␣␣␣5.0000000

4.2 IF 语 句

在实际应用中，除了转移以外，还经常用到条件来进行判断，以确定执行的路径。这就要用到IF语句。

4.2.1 算术IF语句

格式：IF（〈算术表达式〉）L_1，L_2，L_3

作用：根据算术表达式的值转到相应的语句标号去执行。

其中：算术表达式为整、实型算术表达式，式中的变量必须预先赋值。

L_1、L_2、L_3均为可执行语句的语句标号并与算术IF语句在同一个程序单位内。

算术IF语句的执行规则是：

$$\text{若算术表达式的值}\begin{cases}<0, & \text{则转向}L_1\text{。}\\ =0, & \text{则转向}L_2\text{。}\\ >0, & \text{则转向}L_3\text{。}\end{cases}$$

我们来看下面的例子

例3 编写计算函数

$$
Y=\begin{cases} 8, & \text{当 } 0\leqslant X<10 \text{ 时。} \\ 2X^2-10X+0.5, & \text{当 } 10\leqslant X<20 \text{ 时。} \\ 6X+1, & \text{当 } 20\leqslant X<30 \text{ 时。} \end{cases}
$$

的程序。

所给函数有三个分支，且属于三个不同的变化区间。我们用算术IF语句来编写这个程序，实现分支函数的选择。源程序如下：

```
      REAL X,Y
10    READ(*,'(F8.5)') X
      IF(X) 11,12,13
11    GOTO 42
12    Y=8.0
      GOTO 50
13    IF(X-10.0) 12,22,23
22    Y=2.0*X**2-10.0*X+0.5
      GOTO 50
23    IF(X-20.0) 22,32,33
32    Y=6.0*X+1.0
      GOTO 50
33    IF(X-30.0) 32,42,42
50    WRITE(*,100) X,Y
      GOTO 10
42    WRITE(*,'(''ARGUMENT ERROR !'')')
100   FORMAT(1X,'X=',F10.4,4X,'F(X)=',F10.4)
      END
```

运行过程如下：

7.24586↲ （输入数据）

X=7.2459␣␣␣␣F(X)=␣␣␣␣␣8.0000 （输出数据）

−50.639↲ （输入数据）

ARGUMENT ERROR! （输出错误信息）

4.2.2 逻辑IF语句

逻辑IF语句是一种逻辑判断较强的分支程序设计的语句。要用好逻辑IF语句，就必须了解构成逻辑条件的关系运算、逻辑常数、逻辑表达式、逻辑变量以及逻辑运算等。

1.逻辑表达式

逻辑表达式是构成逻辑判别条件的重要组成部分，它产生一个逻辑数据类型的值。

逻辑表达式最简单的形式是：

逻辑常数。

逻辑变量。

逻辑函数。

关系表达式。

以上的简单逻辑表达式可以通过逻辑运算符以及圆括号组合成更为复杂的逻辑表达式。

（1）逻辑常数

逻辑常数只有两个值，即"真"值用 ·TRUE· 表示，"假"值用·FALSE·表示，它们分别占用两个连续字节的存储单元。在内存中表示·TRUE·时，除最末一位为1以外，其余15位全为零；·FALSE·16位均为零。凡是满足逻辑条件者，其值为·TRUE·，否则为·FALSE·。

（2）逻辑变量

在FORTRAN语言中，逻辑常数·TRUE·或·FALSE·可以赋给一个变量，这个变量的数据类型不能是整、实型变量，而只能是逻辑型变量，称为逻辑变量。它用逻辑说明语句定义。

格式：LOGICAL 〈变量表〉

其中变量表是一系列的变量名，若有多个变量名，则彼此之间用逗号“，”隔开。如：

LOGICAL L_1，L_2，AXB

逻辑变量属于局部变量，即只能在定义它的程序单位中引用。

若变量名没有在LOGICAL说明语句中出现，则不能作为逻辑变量引用，即不是逻辑变量。

除用LOGICAL语句明确说明逻辑变量外，还可以用隐含说明语句定义隐含逻辑数据类型。如：

IMPLICIT LOGICAL（A-D）

这说明凡是用字母A、B、C、D开头的变量名均为逻辑变量。

（3）关系表达式

关系表达式用来比较两个算术表达式或两个字符表达式的值，求得的值是一个逻辑值。用两个算术表达式来进行比较的关系表达式称为算术关系表达式。其一般形式是：

〈算术表达式〉〈关系运算符〉〈算术表达式〉

其中：算术表达式可以是整、实型常数、变量以及由此组合而成的算术表达式。

此外，还有字符关系表达式，形式为：

〈字符表达式〉〈关系运算符〉〈字符表达式〉

其中：字符表达式可以是字符常数或字符变量。

这两类关系表达式都可以产生逻辑真或假值。即当指定的关系（条件）成立时，则其值为真，否则为假。

对于字符关系表达式，比较的是字符表达式中的值（即字符）在ASCII码中排列顺序的位置，在前的字符小于在后的字符。若两端字符表达式值的长度不同，则如同用空格在

其后边补充，使它们的长度相同。

关系运算符共有六种：

·LT·	小于	·LE·	小于或等于
·EQ·	等于	·NE·	不等于
·GT·	大于	·GE·	大于或等于

注意：在书写关系运算符时，字母两端必须带上小圆点，如：·LT·，否则出错；关系运算符没有先后运算次序之分。看下面的关系表达式：

A＋X·GE·B＋C　若A＋X≥B＋C，则关系表达式恒为真·TRUE·

K·EQ·K＋1　　K＝K＋1，显然不成立，关系表达式的值为假·FALSE·

'ABC'·EQ·'XYZ'显然不等，其值为假·FALSE·

（4）逻辑运算符

逻辑运算符是构成复杂的逻辑表达式不可缺少的运算符，它可将简单的逻辑表达式（即：逻辑常数、逻辑变量、关系表达式等）连接起来，进行逻辑判断，并产生逻辑值（·TRUE·或·FALSE·）。逻辑运算符有：

·NOT·	逻辑非（反）	最高级
·AND·	逻辑与（乘）	↓
·OR·	逻辑或（加）	最低级

注意：逻辑运算符的两端必须有一个圆点。·AND·和·OR·运算符是二目运算符，·NOT·为一目运算符。

·NOT·为逻辑非（或称逻辑反），其含义是对逻辑值的否定，真值的逻辑非为假，假值的逻辑非为真。

·AND·为逻辑与（或称逻辑乘），运算规则是：当逻辑运算符·AND·两端的逻辑表达式的值均为真·TRUE·时，

则逻辑乘的结果为真·TRUE·。否则结果为假值·FALSE·。

·OR·为逻辑或（即逻辑加），运算规则是：当运算符两端的逻辑表达式的值均为假时，结果为假，其余为真。

在比较复杂的逻辑表达式中，逻辑运算符应按·NOT·、·AND·、·OR·的优先级依次进行运算。

同算术表达式一样，圆括号可改变逻辑表达式的运算顺序。即从最内层圆括号开始，逐渐向外扩展，直到最外层为止，圆括号内的运算仍按其优先级别进行，同级运算则从左向右进行。如果表达式中有算术、关系和逻辑三种运算，其优先级确定为：算术→关系→逻辑

2.逻辑变量的赋值方法及输出

（1）逻辑赋值语句

逻辑赋值语句的作用是将逻辑常数或逻辑表达式的值赋给逻辑变量。

格式：〈逻辑变量〉=〈逻辑表达式〉

作用：将逻辑表达式的值送到逻辑变量中。

（2）用READ语句对逻辑变量赋值

用READ对逻辑变量赋值时要看输入格式rLw，w表示字符个数，r为重复因子。

将Lw格式用于逻辑常数真或假的输入时，可按下列形式书写：

```
      LOGICAL LOGIC1, LOGIC2
      READ(*, 100) LOGIC1, LOGIC2
100   FORMAT(2L8)
```

输入数据为：

␣␣␣␣␣␣␣T␣␣␣␣␣␣␣F↵（T代表真，F代表假）

则变量LOGIC1=·TRUE·，LOGIC2=·FALSE·

输入数据也可以写成：

␣␣·TRUE·␣·FALSE·↵

注意：T和F之前必须有适当个数的空格，而TRUE和FALSE都必须用圆点括起来。

（3）用DATA语句对逻辑变量赋值

我们看下面的示例：

```
      LOGICAL LAB, LAC, L1, L2
      DATA LAB, L1, LAC, L2/2*.TRUE., 2*.
     FALSE./
```

通过数据语句DATA可将两个·TRUE·值分别赋给LAB和L1，将两个·FALSE·值赋给LAC和L2。

（4）逻辑变量的输出

如果需要，也可以将逻辑变量的值输出。如：

```
      WRITE(*, 100)LOG1, LOG2, LOG3
100   FORMAT(1X, 3L7)
```

则输出数据为：

```
␣␣␣␣␣␣T␣␣␣␣␣␣F␣␣␣␣␣␣T
\____L7____/\____L7____/\____L7____/
```

3.逻辑IF语句

格式：IF（〈逻辑表达式〉）st

其中：st为一个除DO、IF、ELSEIF、ELSE、ENDIF、END或其它逻辑IF语句以外的任何可执行语句。

逻辑IF语句是一个可执行语句，它的执行过程是：

（1）首先计算逻辑表达式的值；

（2）若逻辑表达式的值为真，则立即执行该语句中的可执行语句st，然后继续执行IF语句的后继语句。如果列出的

语句为转移语句，则转到指定的可执行语句继续执行。

（3）若逻辑表达式的值为假，则直接执行IF的后继语句，这时的逻辑IF语句相当于一个空语句。

例4 设某班进行期末考试，请统计每个学生和全班的平均成绩。

我们先画出框图，如图4.1所示。

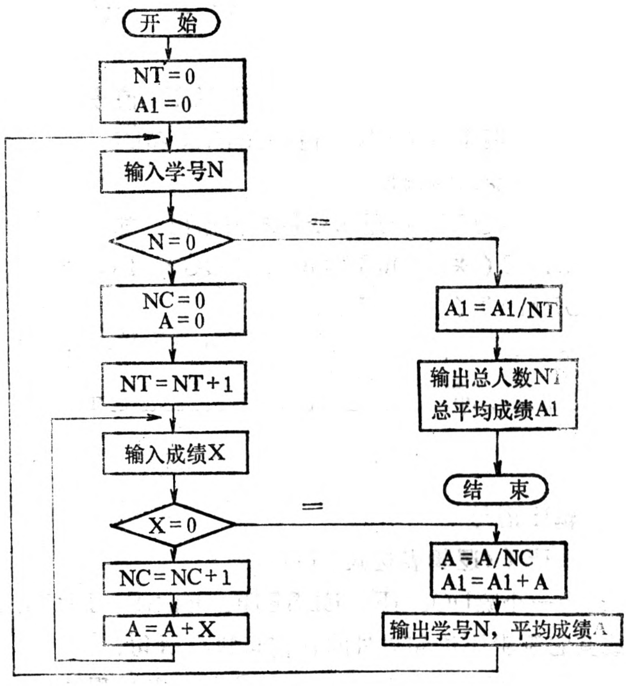


图4.1 学生成绩统计框图

其中各变量代表的意思是：

NT为整型变量，学生人数。

NC为整型变量，该生成绩门数。

N为整型变量，学号。当学号为0时，表示结束。

A1为实型变量，一个学生的总成绩，最后存放该生的平均成绩。

X为实型变量，一个学生的一门功课的成绩。当X＜10^{-8}时，则结束一人的输入并求和与平均成绩。

源程序为：

```
      REAL A1,A,X
      INTEGER NT,NC,N
      NT=0
      A1=0.0
10    READ(*,'(I4)') N
      IF(N.EQ.0) GOTO 110
      NC=0
      A=0.0
      NT=NT+1
20    READ(*,'(F5.1)') X
      IF(X.LT.1E-8) GOTO 50
      NC=NC+1
      A=A+X
      GOTO 20
50    A=A/REAL(NC)
      A1=A1+A
      WRITE(*,100) N,A
      GOTO 10
110   A1=A1/REAL(NT)
      WRITE(*,200) NT,A1
100   FORMAT(1X,'NO=',I5,'   A=',F6.2)
200   FORMAT(1X,'NT=',I5,'   A1=',F6.2)
      END
```

在编写程序时，要处理好逻辑IF语句中的执行语句st与

其后继语句之间的关系，否则将得不到正确的计算结果。

4.3 停 语 句

停语句包括暂停语句和停止语句，它们均属于控制型的可执行语句。在程序中适当的地方设置暂停或停止语句，使程序暂停或中途停止，有利于程序的调试。

1.暂停语句

在程序中设置暂停语句的目的是为了解或检查程序的执行情况，并根据输出的信息决定下一步应采取的措施。在程序的调试阶段往往是必要的。

格式：PAUSE 〔n〕

作用：暂停程序的执行。

其中：待选项n可以是一个字符常量，也可以是不超过5位的无符号整数。

PAUSE语句是一个可执行语句，当程序执行到该语句时，立即处于等待状态，并在屏幕上显示n的值（数字或字符串），或系统所给定的提示信息。当敲入回车键“↲”时，系统又继续运行。

2.停止语句

格式：STOP 〔n〕

作用：停止程序的运行。

这里的n与暂停语句的意义相同。

当程序执行到STOP语句后，则显示出给定的数字或字符串。若没有给出提示参数，则系统输出所规定的信息。

当运行的程序被停止以后，则不能再继续运行。若需要的话，则只能重新执行该程序。

4.4 判定结构程序

判定结构由IF…THEN、ELSE、ELSEIF 和 ENDIF 组成，它有三种判定结构，由这些判定结构可以组成判定结构程序。

1.简单判定结构

简单判定结构由IF…THEN语句开始，以ENDIF语句结束，其间至少嵌入一个可执行的语句。其形式为：

IF（〈表达式〉）THEN

〈可执行语句序列〉

ENDIF

其中：表达式为一个逻辑表达式。

简单判定结构的执行过程如下：

（1）计算逻辑表达式的值；

（2）若逻辑表达式的值为真，则执行指定的可执行语句序列。再执行ENDIF的后继语句。

（3）若逻辑表达式的值为假，则跳过IF和ENDIF之间的可执行语句，直接转到ENDIF的后继语句去执行。

例5 设有数据a_1，a_2，a_3，…an，找出其中的最大数及其序号。试编写该程序。

程序的变量说明：

N为整型变量，表示数据个数。

AMAX为实型变量，存放最大数。

I为整型变量，输入数据计数。

J为整型变量，存放最大数序号。

A为实型变量，存放当前输入的数据。

程序框图如图4.2。

源程序为：

```
      I=1
      J=0
      AMAX=0.0
      READ(*,'(I4)') N
10    READ(*,'(F7.2)') A
      IF(AMAX.LT.A) THEN
        AMAX=A
        J=I
      ENDIF
      IF(I.LT.N) THEN
        I=I+1
        GOTO 10
      ENDIF
      WRITE(*,100) J,AMAX
100   FORMAT(1X,'J=',I4,' AMAX=',F10.2)
      STOP
      END
```

2.基本判定结构

基本判定结构由IF…THEN、ELSE和ENDIF等语句组成，其一般形式是：

IF(〈表达式〉)THEN

〈可执行语句序列1〉

ELSE

〈可执行语句序列2〉

ENDIF

其中的表达式是一个逻辑表达式。其执行过程如下：

(1)计算逻辑表达式的值。

(2)若逻辑表达式的值为真，则执行可执行语句序列

1，然后转到ENDIF的后继语句去执行。

（3）若表达式的值为假，则执行ELSE语句之后的可执行语句序列2，然后再转到ENDIF 的后继语句去执行。

注意：可执行语句序列是一个封闭的语句块，可以从块内转到块外，而不能从块外转到块内。

例6 用判定结构程序求 100 以内的自然数之和。

基本判定部分的程序如下：

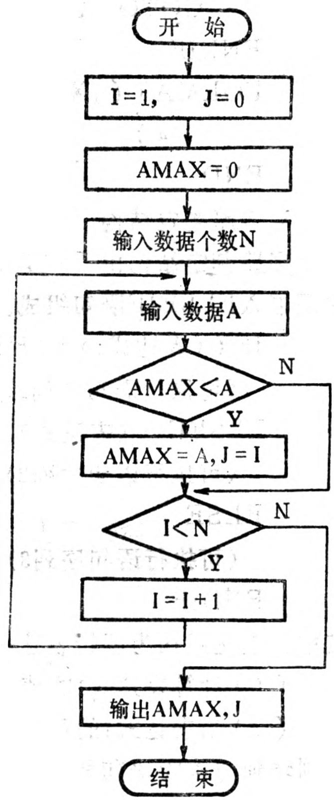


图4.2 求最大数程序框图

```
        ⋮
10      K=K+I
        IF (I·EQ·100)  THEN
          WRITE (*, 100)  K, I
        ELSE
          I=I+1
```

```
      GOTO 10
   ENDIF
100 FORMAT(1X,'1+2+…+100=',I6,'␣␣I=',
  *     I4)
   END
```

3.多路判定结构

多路判定结构由IF…THEN语句开始，以ENDIF 结束，中间嵌入ELSEIF语句组成。它的形式是：

```
IF (〈表达式1〉) THEN
  〈可执行语句序列1〉
ELSEIF (〈表达式2〉) THEN
  〈可执行语句序列2〉
ELSE
  〈可执行语句序列3〉
ENDIF
```

其中：表达式均为逻辑表达式。执行过程如下：

(1)计算表达式1的值。

(2)当表达式1的值为真时，则执行语句序列1，然后控制转到与该IF语句相对应的ENDIF语句的后继语句去执行。

(3)当表达式1的值为假时，则跳过语句序列1而转到ELSEIF语句作进一步判别：

计算表达式2的值。若其值为真，则执行ELSEIF 语句后的语句序列2，然后控制转移到ENDIF语句的后继语句去执行。

若表达式2的值为假，则控制转移到ELSE之后的语句序列3继续执行，直到ENDIF语句结束。

多路判定结构的执行过程如图4.3所示：

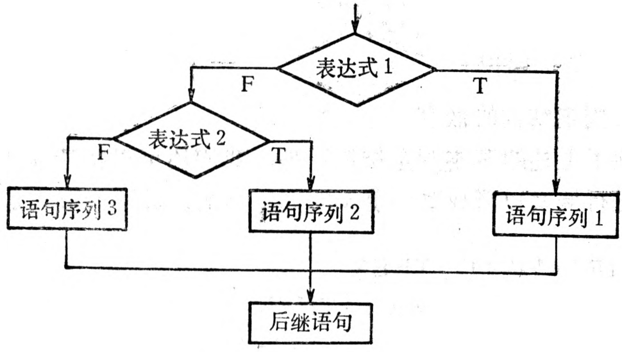


图4.3 多路判定结构流程

例7 计算函数

$$Y=\begin{cases}1 & \text{当}X>0\text{时}\\0 & \text{当}X=0\text{时}\\-1 & \text{当}X<0\text{时}\end{cases}$$

的值，用多路判定结构编写出程序。

源程序如下：

```
INTEGER X,Y
READ(*,100) X
IF(X.GT.0) THEN
  Y=1
ELSE
  IF(X.LT.0) THEN
    Y=-1
  ELSE
    Y=0
  ENDIF
ENDIF
```

```
      WRITE(*,200) X,Y
200   FORMAT(1X,3HX= ,I3,2X,2HY=,I3)
100   FORMAT(I4)
      END
```

4.判定结构的嵌套

除了上述的基本判定结构以外，我们还可以将基本判定结构进行嵌套以形成更为复杂的判定结构。其一般形式是：

```
  ┌ IF (〈表达式1〉) THEN
  │         ┌ IF (〈表达式2〉) THEN
  │         │              ┌ IF (〈表达式3〉) THEN
  │         │              │        ┌ IF (〈表达式4  ) THEN
  │         │              │        │    ⋮
  1         2              3        4  ELSE
  │         │              │        │    ⋮
  │         │              │        └ ENDIF
  │         │              │  FLSE
  │         │              │   ⋮
  │         │              └ ENDIF
  │         │   ELSE
  │         │    ⋮
  │         └ ENDIF
  │  ELSE
  │    ⋮
  └ ENDIF
```

其中的表达式均为逻辑表达式。上列形式示出了四层基本判定结构的嵌套形式，每一层都有一个IF语句和与之匹配的ENDIF语句，而不管其间的语句个数的多少。第4层作为第3层的语句序列被嵌套在第3层中，第3层作为第2层的语句序列被第2层嵌套，其余类推。

当然，也可以在 ELSE 之后嵌套判定结构，使之成为：

```
 ┌ IF (〈表达式1〉) THEN
 │          〈语句序列1〉
 │ ELSE
 │                  ┌ IF (〈表达式2〉) THEN
 │                  │   〈语句序列2〉
 │                  │ ELSEIF (〈表达式3〉) THEN
 │                  │       〈语句序列3〉
1│                 2│       ELSEIF (〈表达式4〉) THEN
 │                  │           〈语句序列4〉
 │                  │       ELSE
 │                  │               〈语句序列5〉
 │                  └ ENDIF
 └ EIDIF
```

其中：各表达式均为逻辑表达式，语句序列为一个或多个可执行语句。

正确使用判定结构可给FORTRAN程序设计带来方便，使程序的结构清晰，易读。反之，则会产生不可预料的错误。因此，使用判定结构来编写程序时必须注意：

（1）判定结构的第一个语句必须是IF…THEN语句，并以ENDIF语句作为该结构的结束语句。如：

```
IF ( X·LT·1000.0 ) THEN
       ⋮
ENDIF
```

这里的IF…THEN与ENDIF匹配。

（2）在IF…THEN、ELSEIF和ELSE 各自的语句序列内的控制转移语句，可在本语句序列内实现转移，但不能转到IF…THEN、ELSEIF和ELSE的语句序列内。

总之，不能从IF…THEN语句序列、ELSFIF语句序列

和ELSE语句序列之外将控制转到语句序列内。

（3）在各种判定结构中的 ELSEIF 语句和ELSE语句均不允许带语句标号。如：

```
         IF（…）THEN
              ⋮
20       ELSEIF（…）THEN
              ⋮
30         ELSE
              ⋮
         ENDIF
```

在结构中出现了带语句标号的ELSEIF和ELSE语句，这是不允许的。

（4）如果有多个IF…THEN、ELSE、ENDIF嵌套在一起，那么总是从最内层的IF…THEN、ELSE、、END-IF开始构成一个正确的判定结构，以此逐步向外扩散，参见例8的源程序。

例8　求a，b，c三个实数中的最大、最小数。程序框图如图4.4。

编写的源程序如下：

```
REAL MAX,MIN
READ(*,'(3F7.2)') A,B,C
IF(A.LT.B) THEN
  IF(B.LT.C) THEN
     MAX=C
     MIN=A
  ELSE
     MAX=B
     IF(A.LT.C) THEN
        MIN=A
```

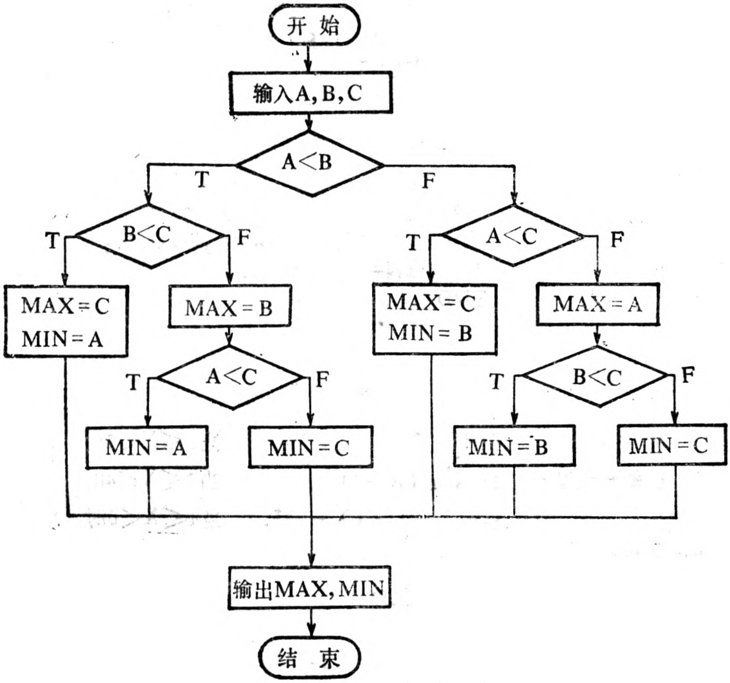


图4.4 求最大最小数程序框图

```
      ELSE
        MIN=C
      ENDIF
    ENDIF
  ELSE
    IF(A.LT.C)THEN
      MAX=C
      MIN=B
    ELSE
```

```
      MAX=A
      IF(B.LT.C) THEN
        MIN=B
      ELSE
        MIN=C
      ENDIF
    . ENDIF
    ENDIF
    WRITE(*,100) MAX,MIN
100 FORMAT(1X,'MAX=',F10.2,
   1'   MIN=',F10.2)
    END
```

习题四

1.编写求 $Y=\begin{cases}(1+X)^2, & \text{当}0\leqslant X<2\text{时}\\ 1+\sin((X-2), & \text{当}2\leqslant X<4\text{时}\\ \sin(X-2)+(X-1)^2, & \text{当}4\leqslant X<6\text{时}\end{cases}$

的值的程序。

2.编写一个FORTRAN程序，求

$$Y=\begin{cases}X^2+\dfrac{1}{X+2}, & \text{当}X\geqslant 1\text{时。}\\ \dfrac{2X}{(X-2)^2}, & \text{当}X<1\text{时。}\end{cases}$$

3.编写一个FORTRAN程序，求

$$1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n}+\cdots,\ \text{当}\frac{1}{n}<10^{-4}\text{时停止。}$$

4.下列判定结构正确吗？为什么？

```
IF(A·GT·B)THEN
     A=X+Y
ELSEIF(A GT·C)THEN
     C=A
```

```
ELSEIF (B·GT·C) THEN
        C=B
        ELSE
        C=X+Y
     ENDIF
ENDIF
```

第五章　循环程序设计

5.1　循环程序的概念

在日常生活中，有许多事件都是在一次又一次地重复着。例如，在一年中有春、夏、秋、冬四个季节，年复一年不断地重复这个过程。

在科技计算中，在很多情况下都可以构成某部分程序的多次重复执行，以获得所需要的结果。简单说来，这种重复过程就叫做循环，用这种方法设计的程序称为循环程序，又称这种结构为循环结构。我们来看看下面的例子。

例1 计算100以内的自然数的和。其程序如下：

```
       I=1
       K=0
 10    IF(I.LE.100) THEN
         K=K+I
         I=I+1
         GOTO 10
       ELSE
         WRITE(*,100) K,I
       ENDIF
 100   FORMAT(1X,'K='I10,'   I=',I10)
       END
```

执行这个程序，当逻辑表达式I·LE·100的值为真时，则执行IF…THEN后面语句序列中的三个可执行语句。直到I·LE·100的值为假，控制转移到ELSE语句执行并输出

计算的结果后，整个程序的运行结束。显然，IF 和 GOTO 语句之间形成了循环。

如果将上述程序改写成如下形式：

```
         I=1
         K=0
10       K=K+I
         IF(I.EQ.100) THEN
           WRITE(*,'(1X,''K='',I10,
        1''  I='',I10)') K,I
         ELSE
           I=I+1
           GOTO 10
         ENDIF
         END
```

显然，IF和GOTO语句之间也形成循环。

上面两个程序都是用基本判定结构结合无条件转移语句来实现程序循环的。而这两个程序却分别代表了两类循环结构。一类称为“当型循环”，即当给定的条件成立时，则执行所要求的循环，直到条件不成立时为止。另一类称为“直到型循环”，即首先进行一次循环，然后再进行条件判断，以确定是否循环。若条件不成立，则将循环控制变量累加一个步长，再继续进行循环。在条件成立时，则终止循环。

通过对例中两个程序执行过程的分析，可以看出循环程序应该具备以下一些特点。

（1）循环程序中必须有控制循环的变量，并在执行循环过程之前对循环变量赋初值。

（2）检测循环控制条件，并根据条件进行循环处理。

（3）构成循环要处理的部分，称为循环体。

（4）循环要有结束之时。

5.2 循环语句

循环程序结构的实现，单用条件判定及控制转移语句是不够的，而且显得很麻烦。检测语句放在循环语句序列的前面或后面是有区别的，而且选择的测试条件也不同。

为了使循环程序结构更容易编写，在FORTRAN语言中设计了专门的循环语句DO。如果用DO循环语句来改写前面的例子，则程序结构就更为清晰了。

```
20      READ(*,'(I3)')N
        F=1.0
        DO 10 I=1, N, 1
            X=1.0/REAL(I)/REAL(I+1)
            F=F+X
10      CONTINUE
        WRITE(*, 100)N, X, F
        GOTO 20
100     FORMAT(1X,'N=', I5,'␣␣X=', F10.6,'
           ␣␣F=', F14.6)
        END
```

源程序中的第三个语句就是循环语句。

1.循环语句的格式

格式：DO L〔,〕I=e_1, e_2〔, e_3〕

其中：DO是循环语句的定义符，标志着循环语句的开始。

L是循环终端语句的标号，它位于DO定义符之后循环控制变量名之前，但它们之间必须用空格隔开，而标号之后也可以用逗号隔开，否则认为是错误的。语句标号的作用就是指明DO循环的范围。

I为循环控制变量，在该FORTRAN标准中，它必须为整型变量。

e_1，e_2和e_3均为整型算术表达式，它们分别为循环初值、终限值和步长（增量）。其中e_3为待选项。如果e_3的值是1且为整型常数时，则可以省略“，e_3”项不写，否则不可省略。

循环变量I后的等号不可缺少，它与算术赋值语句的意义相同。即对循环控制变量I赋值。

2.循环语句的执行过程

一个由循环语句DO和循环语句序列以及终端语句所组成的循环结构的一般形式为：

```
     DO L I=e1, e2, e3
     〈循环体〉
L    〈终端语句〉
```

其中：终端语句通常是循环结构中的最末一个可执行语句。

循环语句的执行过程如下：

（1）首先计算初值、终限值和步长表达式e_1、e_2和e_3的值，将这些值分别保存到循环控制参数M_1、M_2和M_3中。如果步长表达式e_3被省略，则认为其值为1，即循环控制参数M_3置1。这里e_3的值不能为零。

（2）将循环初值赋给控制变量I，即M1中的值赋给整型变量I。

（3）根据循环控制参数M_1、M_2和M_3的值，计算循环次数。计算公式如下：

$$Ic=MAX0(INT((M_2-M_1+M_3)/M_3),0)$$

（4）检测循环次数Ic的值。

若Ic的值为零，则不执行循环范围内的语句，控制转移

到该循环终端语句的下一条语句去执行。

若循环次数Ic>0，则执行循环范围内的全部语句（包括终端语句）。

（5）在循环终端语句执行之后，再进行下列操作：

循环控制变量I增加步长M_3，即：

$I+M_3 \Rightarrow I$

循环次数Ic的值减1，即：

$Ic-1 \Rightarrow Ic$

控制转移到第4步，继续执行。

整个循环的执行过程可用框图表示如下：

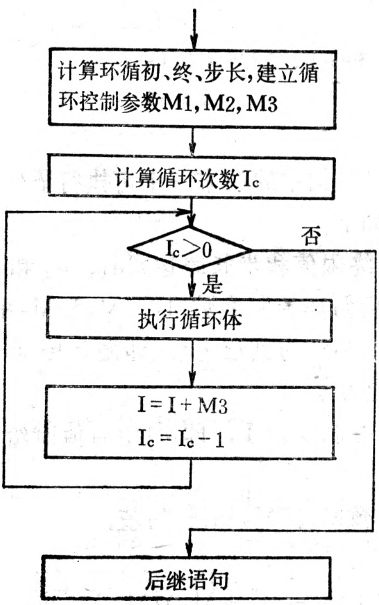


图5.1 循环语句的执行过程

以上的计算、赋值处理过程都是由系统、自动进行，不需人工干予。

例2 设X的值为0.10，0.12，0.15，0.19，求函数

$$f(x)=2x^2+3.5x+0.7$$

的值，编写计算f(x)的源程序。

根据所给数据X的值及f(x)的函数关系，可用循环语句来编写这个程序。在编写程序前，可将f(x)改写成：

$$f(x)=(2X+3.5)X+0.7$$

编写出的源程序如下：

```
      INTEGER I
      REAL F,X
      DO 10 I=1,4
        WRITE(*,'(1X,''INPUT X=?'')')
        READ(*,'(F5.2)') X
        F=(2.0*X+3.5)*X+0.7
10      WRITE(*,100) I,X,F
100   FORMAT(1X,'I=',I2,2X,'X=',
     1F5.2,'  F(X)=',F10.5)
      END
```

执行这个程序时，第一个可执行语句就是循环语句，按循环语句的执行过程，可得：

（1）循环控制参数M1＝1，M2＝4，M3＝1。

（2）循环控制变量I的初值为1，即I＝1。

（3）计算循环次数Ic＝4，即

Ic＝MAX0（（4－1＋1)/1，0）＝4

（4）检测循环次数Ic＞0，则执行循环体，输入X的值，然后计算F的值并输出X、F的值，直到Ic＝0时退出循环。

程序的运行结果如下：

INPUT X＝？

␣0.10↙

I＝␣1␣␣X＝␣␣·10␣␣F（X）＝␣␣␣1·07000

（以下略）

3.对循环语句的限制

FORTRNA标准对循环语句的限制有：

（1）循环控制变量必须为整型变量。初值、终限值和步长必须为整型表达式、整型常数或整型变量。对于整型变

量或整型表达式中的变量必须在语句之前赋值。

（2）循环的终端语句必须是一个可执行的语句。但它不应是无条件GOTO、赋值GOTO、算术IF、IF…THEN、ELSEIF、ELSE、ENDIF、RETURN、STOP、END或DO语句。如果终端语句是逻辑IF语句时，则可包括逻辑IF语句允许的任何可执行语句。

（3）循环控制变量可在循环体内被引用，但不应对它再赋值，否则会破坏循环。同样，也不允许改变循环初值、终值和步长表达式的值，即不允许对表达式中的变量再次定义。

（4）在循环体中，可用控制转移语句从循环体内转到循环体外，但不允许从循环体外转到循环体内。

（5）在DO语句的循环体内，如果出现判定结构语句，则IF…THEN与ENDIF所构成的判定结构程序应全部包含在循环体内。如：

```
              ⋮
┌──→ DO  10  I=1, N
│             ⋮
│        ┌IF(X·LT·1E-3)THEN
│        │    GOTO  20
│        │ELSE
│        │    F=F+X
│        └ENDIF
│10      CONTINUE
└──
20    WRITE(*, 100)N, X, F
```

这个循环结构中的基本判定结构程序整个包含在循环体内，这是正确的。如果改成下面的形式，则是错误的：

```
                ⋮
   ┌──→DO  10  I=1, N
   │     IF (X·LT·1E-3) THEN ──┐
   │       GOTO  20            │
   │     ELSE                  │   判定结构
   └─10    F=F+X               │
         ENDIF ────────────────┘
```

循环体的终端语句在ELSE后的语句序列内，它没有将基本判定结构的全部语句包含在循环范围内，而形成循环结构与判定结构相交叉的形式。这就是错误所在。

（6）如果在判定结构程序的IF块、ELSEIF块或ELSE块中包含有DO语句，则由DO语句所构成的循环结构程序应全部包含在IF块、ELSEIF块或ELSE块中。

下面的判定结构程序是正确的.

```
      IF (X·LT·0.0) THEN
         N=1 ─────────────────┐
         DO  10  I=1, 10      │     IF块
                ⋮             │
10    N=N+1 ──────────────────┘
      ELSEIF (X·GT·0.0) THEN
         N=0 ─────────────────┐
         DO  100  J=1, 100    │     ELSEIF块
100        N=N+J ─────────────┘
         ELSE
           DO  1000 K=10, 1, -1─┐
                 ⋮              │   ELSE块
1000          N=10 ─────────────┘
      ENDIF
```

下面的循环和判定结构程序是错误的：

```
      IF (I·EQ·J) THEN                  ─┐
  ┌──→ DO  99  I=J, J+2                  │ IF块
  │        ⋮                            ─┘
  │   ELSEIF (X·GT·I+1) THEN            ─┐
  │        ⋮                             │
  └─99      M=M+1                        │ ELSEIF块
           ⋮                            ─┘
      ELSE                              ─┐
        M=M+2                            │ ELSE块
           ⋮                            ─┘
      ENDIF
```

这个循环结构的DO语句包含在IF块中，其终端语句又包含在ELSEIF块内，导致跨块交叉，这是不允许的。

5.3　继续语句

格式：CONTINUE

继续语句CONTINUE是一个可执行语句，它的执行不产生任何动作，被称为空语句。CONTINUE语句可用来设置一个语句标号，达到承上启下的目的。在实际使用中，常常将CONTINUE语句作为循环结构的终端语句，特别是循环结构程序的终端语句为控制转移语句时，可在其后加上一个CONTINUE语句并作为循环的终端语句。这样可以避免控制转移发生在终端语句处。另外，为了使 FORTRAN 程序结构清晰、易读，可将源程序分段，也可用CONTINUE语句作为分段的标志。

5.4　循环程序的嵌套结构

前面介绍了DO语句。在那里我们只局限于一个 DO 语

句的情形，这样构成的循环结构，称为简单循环程序，也可以叫做单重循环结构。但应用中有时用单重循环极不方便，常常需要用到多重循环。如：

```
               ⋮
┌──────→ DO 1000 J=1, N
│            SUM=0.0
│    ┌────→ DO 100 I=1, M
│    │           ⋮
│    └─100      CONTINUE
│           SUM=SUM/M
│           WRITE(*, 999)J, SUM
└───1000     CONTINUE
               ⋮
```

1.多重循环程序的执行过程

由于一个单重循环结构可以包含在另一单重循环结构之内，这样就导致多重循环的产生，其结构如下：

```
  ┌──────────── DO 10 I=1, N
  │   ┌──────── DO 20 J=1, M
  │   │  ┌───── DO 30 K=1, L
 1│  2│ 3│          ⋮
  │   │  └─30     CONTINUE
  │   └──── 20    CONTINUE
  └──────── 10    CONTINUE
```

这是一个三重循环结构的形式，它们分别都有自己的终端语句。

多重循环的执行过程，可由单重循环的执行过程推而广之。以下列程序为例，说明多重循环程序的执行过程。

```
          L=0
 ┌──────→DO 10 I=1, 10
 │  ┌───→ DO 20 J=1, 5
 │  │        L=L+1       }内循环体   } 外循环体
 │  └─20   CONTINUE
 └────10  CONTINUE
```

（1）首先执行外循环的DO语句。

建立初值、终值和步长控制参数，它们分别为1、10和1；将初值1赋给循环控制变量I，计算出循环次数为10。测试循环次数，如果循环次数为零，则退出循环，否则执行该循环体。

（2）在执行外循环体时，若遇到内循环，则建立内循环初值、终值和步长控制参数分别为1、5和1，将循环初值1赋给循环变量J，即J＝1，计算出循环次数为5。如果循环次数为零，则退出内循环。如果循次数非零，则执行内循环体直到结束。

（3）执行内循环5次后退出内循环，继续执行外循环直到终端语句。

（4）将外循环控制变量I增加一个步长并将循环次数减1。

（5）如果外循环次数不等于零，则继续执行外循环体并再次进入内循环。即转向第二步开始继续执行。直到结束内循环。

（6）当执行外循环体结束后，若外循环的次数不为零，则继续执行。直到外循环次数为零时，则整个循环结束。

这里的外循环共执行10次，每当执行一次外循环体，则内循环体就要执行5次。因而，内循环体总共执行10×5＝50次。

对于二重循环的执行过程可用框图（图5.2）来表示。

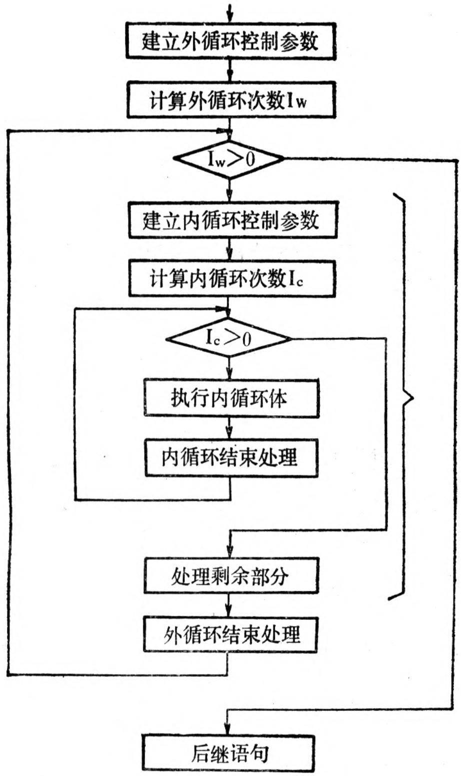


图5.2　二重循环体的执行过程

对于三重以上循环结构的执行过程与二重循环结构的作法基本一样。

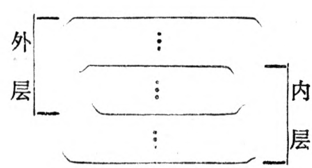
2.对多重循环的限制

（1）内循环必须整个地嵌套在外循环之内，不允许交叉。如下的程序结构是正确的：

```
      DO 10 I=1, 9
        DO 20 J=I, 9
          WRITE(*, 100)I, J, (I*J)
20      CONTINUE
        WRITE(*, '(1X, //)')
10    CONTINUE
100   FORMAT(1X, I2, '␣*␣', I2, '␣=␣'I3)
      END
```

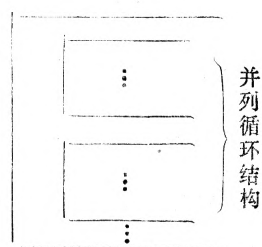
这个程序为二重循环结构，内循环结构已完全嵌套于外循环体内。而下列循环结构是不正确的：

```
      DO 10 I=1, N
        ⋮
        DO 20 J=1, M
        ⋮
10      CONTINUE
        ⋮
20    CONTINUE
```



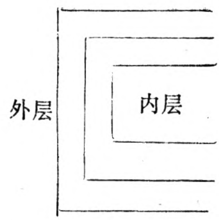
（2）嵌套循环的循环控制变量不能同名，同一层次的多个循环结构（称为并列循环），其循环控制变量可以同名。下列循环结构是正确的：

```
      DO 10 I=1, N
        DO 20 J=1, 5
        ⋮
20      CONTINUE
        DO 30 J=1, 10
        ⋮
30      CONTINUE
        ⋮
10    CONTINUE
```



这是一个二重循环结构，外循环内嵌入了两个循环结构，这两个循环是并列循环，用了同一个循环控制变量。内外循环控制变量是不同的。而：

```
外 ┌──────DO 100 I=1, N
   │  ┌────DO 200 J=1, N
   │  │ ┌──DO 300 I=1, M
   │  │ │内        ⋮
   │  │ └─300   CO TINUE
   │  └──200  CONTINUE
   └──100 CONTINUE
```



是不允许的，从循环结构来看它是正确的，但是外层循环控制变量的名字和内层的同名，所以导致错误的产生。

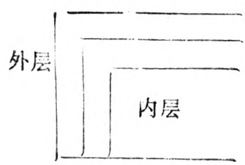
（3）嵌套循环可共用一个终端语句。

在嵌套循环结构中，通常是一个循环结构就有一个终端语句。如果有几个终端语句都是同一语句并连续出现时，则可将它们合并，共用一个终端语句。如：

```
            DO 10 I=1, M
              DO 100 J=1, M
                DO 1000 K=1, N
                     ⋮
1000            CONTINUE
100           CONTINUE
10          CONTINU
```

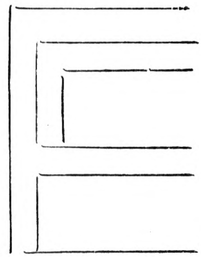
也可写成：

```
            DO 10 I=1, M
              DO 10 J=1, M
                DO 10 K=1, N
                   ⋮
10              CONTINUE
```



下面的程序结构同样是允许的：

```
      DO  10  I=1, M
        DO  200  J=1, N
          DO  200  K=1, N
             ⋮
200      CONTINUE
             ⋮
         DO  10  J=1, N
             ⋮
10     CONTINUE
```



（4）循环的控制转移

循环的控制转移与判定结构的控制转移一样，只能从循环体内转移到循环体外，不允许从循环体外转移到循环体内。

如果共用一个终端语句，则可从嵌套循环的最内层转到该终端语句，其余都不允许转移到该终端语句。

下面再举一个利用循环结构来编写程序的例子。

例3 编写一个程序计算乘积：

$$(y-x_1)(y-x_2)(y-x_3)\cdots(y-x_n)$$

其中：n为项数，计算结果存入变量P并输出。

编写这个程序要输入y和n的值，每循环一次，则输入一个x的值，求y－x的差并与P中的值相乘，可用算术赋值语句p＝p＊(y－x)来实现。这里的p初值应为1.0。源程序如下：

```
      PROGRAM PX
      REAL P,X,Y
      INTEGER N,I
      P=1.0
      READ(*,50) N,Y
      DO 100 I=1,N,2
        READ(*,'(F5.2)') X
        P=P*(Y-X)
```

```
100     CONTINUE
        WRITE(*,10) N,P
50      FORMAT(I3,F5.2)
10      FORMAT(1X,'N=',I4,'   P(X,Y)=',F14.6)
        END
```

执行这个程序的输入数据是.

␣12␣7.65↙

5.45↙

2.37↙

4.56↙

8.92↙

10.53↙

6.38↙

输出结果为：

N=␣␣12␣␣P(X, Y)=␣␣␣␣166.730000

习题五

1.求$1+\frac{1}{2}+\frac{1}{2}\cdot\frac{1}{3}+\frac{1}{3}\cdot\frac{1}{4}+\cdots\frac{1}{n}\cdot\frac{1}{n+1}+\cdots$的值，直到最后一项的值小于$10^{-8}$时为止。

2.抽查某班N个学生期终考试成绩，如下表所示：

学号	政治	语文	数学	英语	化学	物理	平均
2057	85	78.5	95.4	96	88	93	
2058	90	89	100	78.5	69	86	
2059	75	93	67.8	95	78	84	
平均							

要求计算每个人的平均成绩，各科平均成绩及总平均成绩。

3.利用迭代公式：

$$x_{n+1}=x_n+\frac{1}{3}\left(\frac{N}{x_n^2}-x_n\right),$$

计算 $\sqrt[3]{N}$ 的近似值（自选初值 X_0，允许误差ε，N的初值为2，终值为100，步长为1）。

4.计算 $Z=\frac{e^{ax}-e^{-ax}}{2}\sin(x+a)+a\ln\frac{x+a}{2}$ 的值。其中，a的初值为0.10，终限值为0.30，步长0.1；x的初值为1.0，终值为2.0，步长为0.2。

第六章 数 组

FORTRAN程序设计中，除了常用的零星变量和数据以外，还要用到大量的有序数据，这可以用数组进行处理。

6.1 数组的概念

1.数组与数组元素

在FORTRAN语言中，我们可以将多个数据组织起来，按一定的顺序存放在计算机的内存储器中，并为这些数据给出一个统一的名字，这组数据就称之为数组。给定的名字称为数组名。要表示这批数据中的某一个数据，就可以在给出数组名的同时，给出该数据在数组中的序号。如有一张学生成绩统计表：

学号	政治	语文	数学	英语	化学	物理	平均
2057	85	78.5	95.4	96	88	93	
2058	90	89	100	78.5	69	86	
2059	75	93	67.8	95	78	84	
2060	95	67	85.4	80	90	98	

如果将每个学生的成绩都用数组来表示，那么可以建成四个数组。第一个数组的数据表示如下：

（85，78.5，95.4，96，88，93）

现在给这一组数取一个名字，如A1，这就是数组名。要表示其中的某个数据，则可用：

A1（1），A1（2），A1（3），A1（4），A1（5），A1（6）；它们可以分别代表85，78.5，95.4，96，88，93六个数据。这里只用一个下标就可以标识一个数据，因而将这种数组称为一维数组。其余学生的成绩也可以用同样的办法处理。

如果希望将四人的成绩表示成一个数组，并取名为A，那么可用A（I，J）来表示其中第I个人的第J门课程的成绩。如：A（1，3），表示第一个人的第3门课程的成绩（即2057号学生的数学成绩95.4）。由于确定一个成绩需要两个下标（行、列），因而这种数组称为二维数组。

归纳起来说，凡是一组有序的数据都可以用数组的形式表示出来。

2.数组的维数与下标变量

从前面的简单介绍中已经看到：使用一个数组元素时，必须给出它在数组中的顺序号，这个顺序号称为下标。因此又称数组元素为下标变量。例如：100个整数可以表示为一个数组S，它共有100个元素，即S（1），S（2），S（3），…，S（100）。这些S（1），S（2），…，S（100）分别都是数组S的下标变量。

若令I＝90，则S（I＋10）即表示下标变量S（100）。由此可见，下标变量中除了用整型常数来表示元素的序号外，还可引用整型表达式的值来表示元素的序号。但是出现在该表达式中的整型变量必须预先赋值，这种表达式称为下标表达式。

在下标变量中出现的下标表达式的个数称为数组的维

数。如S（2）只出现了一个下标表达式，这说明该数组S是一个一维数组。

在下标变量中若出现两个下标表达式，如下标变量A(1，2)和A(I，J)，则称该数组为二维数组。

FORTRAN标准规定：数组的最大维数是3，也就是说，在下标变量中最多可以出现三个下标表达式。如果超过三个下标，则认为是错误的。

在同一个程序单位中，同一个数组的下标变量的下标表达式的个数必须相同。如下标变量A(1，2)和A(M+1,J)，是正确的，而A（3）是错误的。

下标变量可以出现在：

（1）表达式中，被引用之前必须预先赋值。

（2）赋值语句的左端，该下标变量则被重新赋值。

（3）DATA语句中并通过DATA语句赋给初值。

（4）输入输出语句中，实现数组元素的输入输出。

数值名也可以出现在DATA语句中并通过DATA语句对数组的每一个元素赋值；也可以在输入输出语句中作为输入输出对象，以输入或输出该数组的每一个元素值。

6.2 数组的定义

数组可通过两种方式进行定义。

6.2.1 数组说明语句

数组的定义方式之一，即DIMENSION语句。DIMENSION语句又被称为维语句，它用于指明数组名和数组元素的最大数目，并分配相应的内存空间。

格式：DIMENSION 〈数组名〉(d_1[，d_2][，d_3])，…

作用：定义一个或多个数组。

其中：数组名就是为有序数的全体取的名字；d_1、d_2、d_3为维说明符，维说明符最多只允许出现3个。di(i=1，2，3)的值确定了相应元素序号变化的最大范围，称为上限，其下限总是1。

常用的维说明符di(i=1，2，3)，可以是无符号型常数，整型变量和“*”号。若维说明符是一个整型常数，它指明数组在该维上元素的最大值；若各维都是整型常数，则该数组的容量是不变的。若维说明符是整型变量，那么，这个数组说明符只能出现在子程序中，其中表示维说明符的整型变量的值必须在进入该子程序之前对其赋初值。这样的数组被称为可调数组，即其大小随整型变量值的改变而改变。若维说明符是一个“*”号，则该数组是假定尺寸数组，其维界不确定。参见下面的例子：

```
DIMENSION A(4), B(3, 3)
DIMENSION C(4, 3), D(2, 10)
```

这里定义了A、B、C、D四个数组。A是一维数组，它共有四个元素；B是二维数组，其第一维上界是3，第二维上界也是3，B数组共有9个元素。其余的数组C和D与此类似。

使用DIMENSION语句来定义数组时应注意下列几点：

(1)在一个程序单位中可使用多个DIMENSION语句来定义多个数组，但数组名不能相同。

(2)用DIMENSION语句定义的数组，其类型遵循隐含规则。也就是说，在DIMENSION语句中定义的数组，只有整型和实型两种类型。

（3）在主程序单位中，DIMENSION语句定义的数组的维说明符只能是整型常数，称这种形式的数组为常界数组。可调维数组只能在子程序中定义。如：

```
PROGRAM MAXT
DIMENSION A(10, 10), B(20), C(50)
DIMENSION D(2, 3), M1(10)
```

是正确的，而：

```
PROGRAM MAB
DIMENSION IM(K, 10)
```

是不正确的，因为它是主程序单位的说明部分。在数组说明符中，第一维上界为整型变量K，因而IM是一个可调数组，它只能在子程序中出现。

（4）数组的大小

数组的大小等于数组中元素的个数，即各维上限的乘积。

6.2.2 用类型语句定义数组

在上节中用到的DIMENSION语句，除了用于数组的定义、维说明以及维上界之外，还提供了数组的类型。用DIMENSION语句只能定义整型或实型数组。因此，对数组的应用带来了一定的局限。

为了更广泛地引用数组这个有效的数据处理手段，FORTRAN语言还允许用类型语句来定义数组。具体作法是：

（1）将数组名当作变量在类型语句中定义其类型，然后再用DIMENSION语句定义为数组。如：

```
INTEGER  A, B, IN, AB
DIMENSION  A(10), B(2, 2), IN(2, 2, 2)
```

这里定义了四个整型变量A、B、IN和AB，而在维语句中又

对A、B和IN进行了再定义，并给出了每一维的上界和维数。其中A为10个元素的一维整型数组；B为四个元素的二维整型数组；IN为8个元素的三维整型数组。

（2）将数组说明符直接置于类型语句中，使之变得更简洁。如：

```
REAL IA(10), B(2, 2)
INTEGER A(24, 24)
CHARACTER CHAR1(10)*3
LOGICAL L(2, 3)
```

这里说明了四种类型的数组。IA和B为实型数组，分别为一维10个元素和二维4个元素；数组A为具有24×24个元素的二维整型数组；L为具有6个元素的二维逻辑型数组；CHAR1为一维字符型数组，共有10个元素，每个数组元素可存放3个字符。

6.3 数组的存储与输入输出

6.3.1 数组元素的存储顺序

一个数组通过维语句或类型语句来定义。数组一旦被定义，它就能够提供名字、类型、维数、维的大小以及数组的元素个数等信息。但是，它的全部元素是如何存放的呢，即内存储单元是如何分配的呢？

FORTRAN77语言标准规定，数组中的数组元素以“按列排列”的顺序连续分配存储单元。

对于一维数组，则按下标的值增加的顺序分配存储单元。

对于二维数组，数组元素中最左边的下标最先变化，然

后第二个下标再变化…。即数组元素的第二个下标从下界1到上界(从小到大)每固定一次，第一个下标都从下界变化到上界。直到第一维和第二维的下标都变化到上界时为止。如：

DIMENSION A(4, 4)

定义了一个二维的实型数组A，第一、二维大小均为4，它的数组元素共16个。在内存储器中的分配如下：

1	A(1,1)	5	A(1,2)	9	A(1,3)	13	A(1,4)
2	A(2,1)	6	A(2,2)	10	A(2,3)	14	A(2,4)
3	A(3,1)	7	A(3,2)	11	A(3,3)	15	A(3,4)
4	A(4,1)	8	A(4,2)	12	A(4,3)	16	A(4,4)

表格中下标变量之前的数字1、2、…15、16为该变量在内存单元中的顺序号。由顺序号可以看出下标值的变化情况。三维数组的下标变化情况与此类似。

实际上，数组中的每一个元素都要分配一个顺序号，如何根据数组元素的下标来计算元素的位置顺序号呢？计算公式为：

一维数组元素序号 $=(L_1-1)+1$

二维数组元素序号 $=(L_2-1)*d_1+(L_1-1)+1$

三维数组元素序号 $=(L_3-1)*d_1*d_2+(L_2-1)*d_1+(L_1-1)+1$

其中：L_1、L_2、L_3分别为数组元素第一、二、三维下标表达式之值。

d_1、d_2、d_3分别为第一、二、三维的维上界。

例如：REAL M(2, 2, 3)语句定义了一个三维实型数组M。数组元素M(2, 1, 3)的顺序号为10，即：M(2, 1,

3）的分配序号＝（3－1）＊2＊2＋(1－1）＊2＋(2－1)＋1

＝8＋1＋1＝10

6.3.2 数组的输入输出

数组定义以后，数组的每一个元素可作为下标变量而被引用。它可以出现在表达式中作为已赋值的运算对象，也可以作为赋值语句的左端变量，为了得到数组和数组元素的结果，还可以出现在输入输出语句中，以便输入或输出数组的值。

1.数组元素的输入与输出

数组元素的输入或输出与普通变量的输入或输出是一样的，而整个数组元素的输入或输出通常与循环语句联用。

例1 数组元素的输入。

```
      REAL A(5,2),B(5,2)
      DO 20 J=1,2
        DO 20 I=1,5
          READ(*,'(F6.4)') A(I,J)
20    CONTINUE
      DO 10 J=1,2
        SUM=0.0
        DO 40 I=1,5
          SUM=SUM+A(I,J)
          WRITE(*,100) A(I,J)
40      CONTINUE
        WRITE(*,'(''SUM='',F10.4)') SUM
10    CONTINUE
100   FORMAT(F8.4)
      END
```

当执行这个程序时，通过循环语句将输入数据赋给数组A的各个元素，这里输入数据的顺序与A数组在内存储单元

中的存放顺序要一致。

2.用数组名作输入输出对象

除了输入或输出数组元素以外，也可以直接用数组名来进行输入或输出。如：

```
      REAL DME(3, 2)
      INTEGER AME(3, 3)
      READ(*, '(6F4.1, /, 9I3)') DME, AME
            ⋮
```

执行该程序时，若敲入数据：

```
␣1.1␣1.2␣1.3␣1.4␣1.5␣1.6
␣10␣20␣30␣40␣50␣60␣70␣80␣90↙
```

就把这些数据按内存分配的顺序依次赋给了各个数组元素。当然，也可以一列一列地输入数据。

例2 数组元素的输入顺序

```
      REAL DME(3, 2)
      INTEGER AME(3, 3)
      READ(*, 100)DME
100   FORMAT(3F4.1)
      READ (*, 200)AME
200   FORMAT(3I4)
        ⋮
```

执行前一个读语句时，先输入数组的第一列，然后再输入第二列，即：

```
␣1.1␣1.2␣1.3↙
␣1.4␣1.5␣1.6↙
```

执行后一读语句时的情形与此类似：

```
␣␣10␣␣20␣␣30↙
␣␣40␣␣50␣␣60↙
```

␣␣70␣␣80␣␣90↙

也可以用数组名直接作为输出对象，与输入时类似。如果整型数的元素分别为1，2，…，9，那么下列程序：

```
          INTEGER  AB(3, 3)
               ⋮
          WRITE(*, 1000)AB
    1000  FORMAT(1X, 3I5)
               ⋮
```

执行后输出数据是：

␣␣␣␣1␣␣␣␣2␣␣␣␣3

4	5	6
7	8	9

3.隐循环输入输出

在输入输出语句的输入输出表中使用数组元素或数组名，对数组元素的赋值或输出来说，已经够方便了。而FORTRAN又提供了一种隐循环的输入输出方式。

我们先来分析下列循环程序的执行情况：

```
        DO 10 I=1, 10
         WRITE(*,'(I2)')A(I)
     10 CONTINUE
```

假设数组A是一个已赋值的一维整型数组，共有10个数组元素，其元素值分别为1，2，…，10。执行循环语句时，将会把A(1)，A(2)，…，A,10)10个数据顺序输出。

若引入隐循环形式，则上述循环程序将变得更为简单：

```
          WRITE(*, 100)(A(I), I=1, 10)
    100   FORMAT(1X, 10I4)
```

执行WRITE语句时，将I赋初值为1，计算出隐循环的循环次数为10，检测次数是否为零，若为零，则结束隐循环，否则开始循环，输出第1个元素的值。然后I增加步长1，

再执行隐循环，输出第二个数组元素的值，直到数组元素输完为止。

（1）隐循环的一般形式

格式：（IOLIST，I=e_1，e_2〔，e_3〕）

其中：IOLIST为输入输出对象表；I为隐循环控制变量；e_1，e_2和e_3分别为隐循环的初值、终限值和步长，循环次数由e_1、e_2和e_3决定。

实际上，隐循环与DO语句的循环方式完全相同，其要求以及执行过程可参考DO语句的说明。

隐循环主要用于输入输出语句。

例3　隐循环的用法。

```
      PROGRAM FX
      DIMENSION X(4),A(3),F(4)
      READ(*,'(4F5.2)') (X(I),I=1,4)
      READ(*,'(3F4.1)') (A(I),I=1,3)
      DO 10 I=1,4
        F(I)=A(1)
        DO 20 J=2,3
           F(I)=F(I)*X(I)+A(J)
20      CONTINUE
10    CONTINUE
      WRITE(*,30)(I,X(I),I,F(I),I=1,4)
30    FORMAT(1X,'X(',I2,' )=',F5.2,
     1'  F(',I2,' )=',F7.4)
      END
```

执行这个程序时，输入数据由两个READ语句承担，其输入对象均为隐循环。

当执行到输出语句WRITE时，则四个函数已被计算完毕，在输出对象表中也用到了隐循环，它重复执行了四次，即输出四个数据。

（2）隐循环的嵌套

隐循环也和DO循环语句一样可以从内层向外层扩展，但不存在任何形式的控制转移。隐循环的嵌套最适合于多维数组元素的输入或输出。

设二维数组A在内存的值为：

$$\begin{pmatrix} 1.0 & 1.0 & 1.0 & 1.0 \\ 2.0 & 1.0 & 1.0 & 1.0 \\ 3.0 & 2.0 & 1.0 & 1.0 \\ 4.0 & 3.0 & 2.0 & 1.0 \end{pmatrix}$$

由下列程序可输出4×4列的方形图。

```
    ⋮
WRITE(*,'(4F4.0)') ((A(I,J),J=1,4),I=1,4)
    ⋮
```

其中 (A(I,J),J=1,4) 为内隐循环，((A(I,J),J=1,4),I=1,4) 为外隐循环。

实际上，这个输出语句可认为是由语句：

```
      DO  10  I=1, 4
10       WRITE(*,'(4F4.0)')(A(I,J), J=
         1, 4)
```

改写而成的。

（3）对变量的隐循环

除了对数组元素的输入或输出可以用隐循环以外，普通变量的输出也可以用隐循环。

如果要求重复变量X、Y和Z的值，则可用下列隐循环方式来实现：

```
      WRITE(*, 10)(X, Y, Z, I=1, 2)
10    FORMAT(1X, 6F6.2)
```

如果X=10.5，Y=7.82，Z=20.75，则输出数据为：

␣10.50 ␣␣7.82 ␣20.75 ␣10.50 ␣␣7.82 ␣20.75

X Y Z X Y Z

利用对数组的输入方法，可以对数组赋初值。如果对数组赋同一个初值（如：零），那么用DATA语句最为简单。如：

```
INTEGER A(6, 6)
DATA A/36*0/
```

又如：

```
REAL A1(10), IB(5), N
CHARACTER CH(4, 5)*2
DATA N, A1, IB/1.3, 10*1.5, 5*100.789/
DATA CH/20*'␣A'/
```

这里的DATA语句将1.3赋给N，一维实型数组A1的值均为1.5，而IB的各元素值则为100.789；CH为字符型数组，共有20个元素，每个元素可存放2个字符，DATA语句中将字符串'␣A'赋给CH的全部元素。

6.4 数组的应用实例

例4 统计学生成绩并将个人平均成绩按从大到小的顺序输出。学生成绩如下（假设共8行）：

学号	政治	语文	数学	物理	化学	英语	总成绩	平均成绩
3041	94	89.7	99	88	100	97.3		
3042	75.6	87	100	65.7	75.8	80		
3043	85	91.2	63.5	83.8	68.4	90.5		
…	…	…	…	…	…	…		

解题过程.

第一步，计算每个学生的总成绩并求出平均成绩。

第二步，以平均成绩为准排序，并进行对应数据的交换。即找出平均成绩最高者与表中第一行对应元素进行交换。

第三步，将排好次序的数据全部输出。

程序的框图（图6，1，见110页）及编写的FORTRAN源程序如下：

```
      PROGRAM EV
      DIMENSION STUDEN(8,9)
      WRITE(*,'(''INPUT N=?'')')
      READ(*,'(I3)') N
      READ(*,'(7F6.1)') ((STUDEN(I,J),
     1J=1,7),I=1,N)
      DO 100 I=1,N
         SUM=0.0
         DO 200 J=2,7
            SUM=SUM+STUDEN(I,J)
200      CONTINUE
         STUDEN(I,8)=SUM
         STUDEN(I,9)=SUM/6.0
100   CONTINUE
C
C
C
      DO 20 I=1,N-1
         B=STUDEN(I,9)
         K=I
         DO 10 J=I+1,N
            IF(STUDEN(J,9).GT.B) THEN
              B=STUDEN(J,9)
              K=J
            ENDIF
```

```
10          CONTINUE
            DO 30 J=9,1,-1
               B=STUDEN(K,J)
               STUDEN(K,J)=STUDEN(I,J)
               STUDEN(I,J)=B
30          CONTINUE
20        CONTINUE
          WRITE(*,40) (STUDEN(I,1),
         1(STUDEN(I,J+1),J=1,6),
         2STUDEN(I,8),STUDEN(I,9),I=1,N)
40        FORMAT(F6.0,6F6.1,F6.1,F6.1,/)
          END
```

在这个程序中的

```
      DO 30 J=9, 1, -1
            ⋮
30    CONTINUE
```

循环结构是对最大平均值所在行与第I行的对应元素值进行交换。

习　题　六

1.生成下列数组

$$A=\begin{pmatrix} 1 & 1 & 1 & 5 \\ 2 & 1 & 1 & 6 \\ 3 & 2 & 2 & 7 \\ 4 & 3 & 2 & 8 \end{pmatrix}$$

2.编写求两个矩阵相乘的程序。

3.求多项式 $Pn(x)\ a_0x^n+a_1x^{n-1}+\cdots+a_{n-1}x+a_n$ 的和。

4.对于X=1.00, 1.01, 1.02, …, 3.00, 求

$$Y=41.29\sqrt{1+X^2}+X^{\frac{1}{3}}e^x$$

试编其程序。

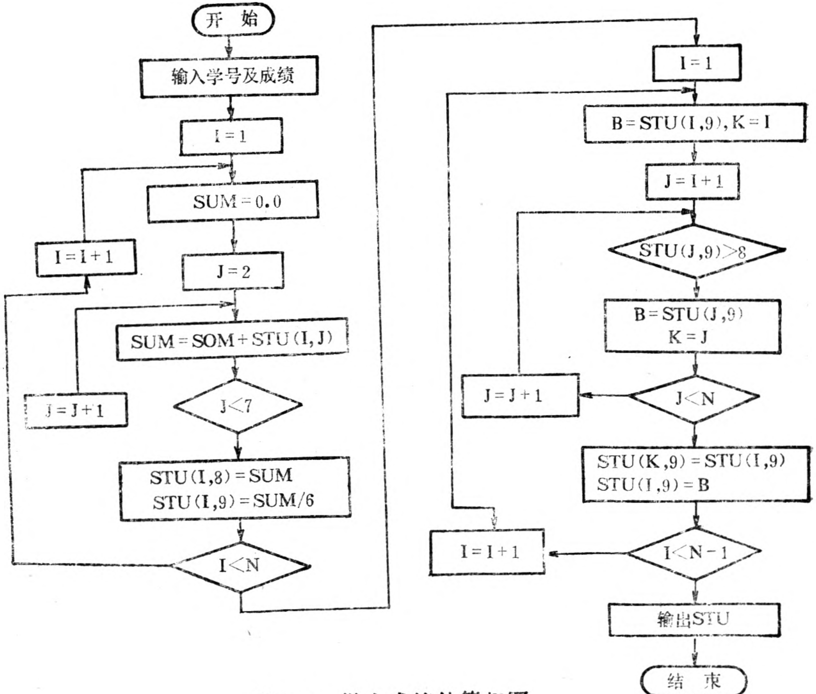


图6.1 学生成绩处管框图

第七章　函数与子程序

在前面的章节中，我们已经介绍了简单程序（即顺序结构）、分支程序以及循环程序等知识，掌握了一定的编程技术。不管是用那种结构来设计程序，都局限于一个主程序单位。而通过说明语句定义的各种类型的FORTRAN名都局限于该程序单位。

本章将介绍函数、子程序以及有关程序单位之间如何进行信息传递，这将使你的程序设计更加丰富多彩。

7.1　函　数

在FORTRAN语言中的函数可分为：内部函数、语句函数以及外部函数。

7.1.1　内部函数

在第一章中，已经对内部函数作过简单介绍并列出常用的一些内部函数。下面将进一步介绍内部函数及其有关内容。

内部函数也称标准函数，它由系统预先定义，可在程序中直接使用。所有内部函数名、定义、参数个数和类型请参见附录。

使用内部函数时必须注意以下几点：

（1）IMPLICIT语句不能改变一个内部函数的类型。

（2）对于允许多个自变量的内部函数，在使用中所有变量的类型必须相同。

（3）内部函数名可出现在INTRINSIC内部语句中。

（4）内部函数的自变量必须满足给定的条件，否则将引起运行错误。

（5）若自变量为角度时，则必须转换为弧度值。

7.1.2 语句函数

FORTRAN语言中还可以定义变量之间的函数关系，这种函数一经定义，就可以象标准函数一样的使用。这就是语句函数的作用。

1.语句函数语句

格式：$F(X_1, X_2, \cdots, Xn)$=〈表达式〉

其中：F表示语句函数的名字，它应遵循FORTRAN语言的命名规则。语句函数的类型可用类型语句或I-N规则来定义。若用类型语句说明，则必须将类型语句放在函数语句之前。

圆括号中的X_1，X_2，…，Xn为语句函数的形式参数（或称哑元），形参彼此之间不允许同名，也不允许为常数、数组或数组元素。其类型可用隐含规则定义，也可出现在类型语句中，但必须在该语句函数语句之前。如果形式参数与本程序单位中的变量同名时，则其类型与变量同类型。

符号右端的表达式可以是算术表达式或逻辑表达式等。在表达式中除了形式参数X_1，X_2，…，Xn以外，还可以有常数、变量、内部函数和本程序单位中已经定义过的语句函数等。

语句函数语句不是可执行语句，而是一个特殊的说明语句，它可以出现在程序单位中的所有说明语句之后和第一个

可执行语句之前。

语句函数语句不象可执行语句那样按顺序执行，由它定义的语句函数则只能在表达式中引用。

例1 将角度化成弧度，再求函数值。假设角度 $Dn=X_1°X_2'X_3''$，转换公式为：

$Dn=\frac{\pi}{180}(X_1+X_2/60+X_3/3600)$。源程序如下：

```
      PROGRAM COUNTA
      F(X1,X2,X3)=3.1415926535*
     1(X1+X2/60.0+X3/3600.0)/180.0
      A=F(36.0,43.0,27.0)
      B=F(15.0,24.0,0.0)
      C=F(8.0,16.0,54.0)
      S=COS(B)**2-4.0*SIN(A)*TAN(C)
      WRITE(*,100) A,B,C,S
100   FORMAT(1X,'A=',F10.7,'  B=',
     1F10.7,'  C=',F10,7/1X,'S=',F10.7)
      END
```

这个程序中定义了一个语句函数语句：

F(X1, X2, X3)=3.14159265*(X1+X2/60.0+X3/3600.0)/180.0

它的函数类型是实型、具有三个参数X1、X2和X3并且均为实型。在这个程序单位中，计算函数值S时，引用了sin(A)、cos(B)和tan(C)。因此，在引用内部函数之前就必须将其度、分、秒数据分别转换为弧度值，所以程序中先后共引用了三次语句函数F，从而获得A、B和C的值为弧度值。执行该程序的结果为

A=␣␣.6409580␣␣B=␣␣.2687810␣␣C=␣␣.

1445420

S=␣␣·5813280

2.关于语句函数的一些规定：

（1）一个语句函数只能用一个语句定义完，如果表达式太长，可以在续行中书写。

（2）语句函数只有一个函数值。

（3）语句函数的右端表达式中可以出现已定义的语句函数或引用内部函数，但不能出现该语句函数自己。

（4）语句函数的形式参数不允许为数组或下标变量。

（5）语句函数名属于局部变量，只能在定义它的程序单位中引用。

（6）语句函数可以没有形式参数X_1，X_2，…，X_n，称这种形式的语句函数为无参语句函数，但圆括号不能省略。其形式为：

F（　）=〈表达式〉

3.语句函数的引用

语句函数的引用与内部函数的形式一样，即：

〈语句函数名〉（〔〈实在参数表〉〕）

使用中应注意：

若为多个实在参数，则彼此之间必须用逗号分隔，并且实在参数的个数、顺序、类型必须与形参一致。

变量名、下标变量名、函数名（内部函数、语句函数）等都可以作为实在参数。

7.2　函数子程序

我们知道，语句函数是由一个语句函数语句定义的；它基

于一个表达式；通过语句函数语句定义的函数名，只局限于一个程序单位的内部；语句函数的函数值只有一个；它的形参只允许是普通变量名；由于语句函数名是局部性的，所以，只能在定义该函数的程序单位中引用，退出定义它的程序单位后，则不再有意义，除非被再次定义；形式参数的限制，使得数组名、下标变量名都不能作语句函数的形式参数。由于这些，使语句函数的应用范围受到限制。为了使程序设计更为方便，FORTRAN提供了定义外部函数的方法来定义函数子程序，从而扩大了函数的应用范围。

7.2.1 函数子程序的定义

函数子程序的结构形式是：

```
〔〈类型〉〕 FUNCTION FUN(〔〈形式参数表〉〕)
        〈说明语句部分〉      ┐
             ⋮               │
        FUN＝〈表达式〉      │
             ⋮               ├  函数子程序体
        RETURN               │
             ⋮               │
        END                  ┘
```

其中：

〔〈类型〉〕 FUNCTION FUN(〔〈形式参数表〉〕)称为FUNCTION语句，也可以叫做(外部)函数语句。它指明一个程序单位是函数，并给出其类型、名称和形式参数。

选择项〈类型〉用于给出函数名FUN的类型，它可以是下列类型定义符之一：INTEGER、REAL或LOGICAL等。

FUNCTION是函数语句的语句定义符，它标志着一个

函数子程序的开始。

FUN是函数子程序名，它是一个全局名，其命名规则如同普通FORTRAN名。FUN具有类型，若在FUNCTION的前面未出现〈类型〉，则函数类型遵循I-N规则，否则由类型语句INTEGER、REAL或LOGICAL定义，或者由IMPLICIT指定的隐含类型来确定，也可以在该函数单位内用类型语句来定义。

形式参数表为待选项，可根据要求设置或不设置形式参数。如果没有形式参数，函数名后也必须保留圆括号，其形式变成：

〔〈类型〉〕 FUNCTION FUN（ ）

在函数语句之后到END语句之间的语句序列组成了函数子程序体。其中：

说明语句部分的语句顺序同主程序单位是一致的。外部函数的类型、形式参数的类型、以及该函数单位内部引用的语句函数、变量和数组的类型，均可以在说明部分指定，其说明方式也同主程序中一样。

在函数子程序中必须有至少一个赋值语句对函数名FUN赋值，以保存求得的函数值供返回之用。

另外，还要有返回语句RETURN。

这是一个可执行的语句，它结束函数子程序使之返回到引用或调用的地方。若RETURN语句的下一个是END语句时，则RETURN语句可以省略，这时的END语句与返回语句RETURN等效。反之，则不能省略END语句，因为它是作为程序单位结束的唯一语句。

例2 计算当X＝1.5时，多项式

$$P_5(X)=3.5X^5+2.1X^4+4.15X+1.5$$ 和

$$P_3(X)=2.4X^3+1.8X^2+0.75X+4.0$$

的值，用函数子程序的结构编写其计算程序。

这里将多项式的计算过程统一编写成一个函数子程序的结构，因此，可以很容易地写出计算多项式的函数子程序：

```
      FUNCTIUN FN)·N,A,X)
      DIMENSION A(7)
      S=A(1)
      DO 20 I=2,N+1
        S=S*X+A(I)
20    CONTINUE
      PNX=S
      RETURN
      END
```

程序中通过FUNCTION语句定义了一个名为PNX的实型函数，它具有三个形式参数N，A，X。其中N为整型，代表多项式的次数；A为实型数组，为计算时提供多项式系数；X为实型，是多项式的自变量。

要计算$P_5(1.5)$和$P_3(1.5)$的多项式值，只需要引用两次PNX函数子程序即可。

7.2.2 函数子程序的引用形式

1.函数子程序的引用形式

函数子程序的引用与语句函数和内部函数一样，都在表达式中引用，它可出现在表达式的任何地方，并将这个函数值用于该表达式的运算中。

函数子程序的引用形式是：

$$F(a_1, a_2, \cdots, a_n)$$

其中：F为已定义的（外部）函数名。

$a_1, a_2, \cdots, a_n$是实在参数表，实在参数可以是表达

式、普通变量、下标变量、数组名、内部函数引用、函数引用以及函数名和子程序名等（注意：不允许子程序名作形参和实参）。

如果定义的函数子程序没有形式参数，其引用为

F（　　）

的形式。

2.函数子程序的执行过程

（1）若实在参数为表达式时，则先对表达式求值。

（2）实在参数与形式参数的结合。

实在参数与形式参数的结合称为形实结合（或叫做哑实结合）。

若实在参数为常数或表达式求值得到的值，称这种形式的结合为值的结合。

若实在参数为已赋值的变量（包括下标变量）、数组名时，这种形式的结合为名的结合。

（3）经过形实结合，形式参数获得了所需要的值后立即执行函数子程序，直到遇到RETURN语句或END语句时，则将控制返回到引用的地方，并带回所计算的函数值参加表达式的运算。

例3　对于例2的子程序，其主程序为：

```
      PROGRAM P64
      DIMENSION A(6),B(4)
      WRITE(*,'(''INPUT A(I)=?(I=1,6);'',
     1''B(I)=?(I=1,4)'')')
      READ(*,5) (A(I),I=1,6),(B(I),I=1,4)
      P5=PNX(5,A,1.5)
      P3=PNX(3,B,1.5)
      WRITE(*,10) X,P5,X,P3
5     FORMAT(6F5.2,4F5.2)
```

```
10     FORMAT(1X,'P5(',F4.1,' )=',F10.7,
     1 ' P3(',F4.1,' )=',F10.7)
       END
```

执行由多个程序单位组成的可执行程序时，系统总是从主程序开始执行，在上述主程序中。引用了两次PNX函数子程序。当执行第4个语句后，则将P_5(x)和P_3(x)的系数分别读入到实型数组A和B中，其对应元素值为(3.5，2.1，0.0，0.0，4.15，1.5)和(2.4，1.8，0.75，4.0)。当执行算术赋值语句时，则右端表达式中遇到引用函数子程序PNX，而实在参数为5，A，1.5。其中的A为数组并对应PNX中的形式数组A，5和1.5分别对应于形式参数N和X。由于引用PNX的第一个和第三个实在参数均为整型和实型常数，所以，直接将5和1.5分别赋给形式参数N和X。第二个实在参数为6个元素的一维数组A，而在PNX中的形式数组的长度能容纳A，因此可进行名的结合。形实参数结合完毕后则立即执行PNX，此时相当于执行下列程序块：

```
      S=A(1)
      DO 20 I=2, 6
        S=S*1.5+A(I)
20    CONTINUE
      PNX=S
      RETURN
```

当执行到RETURN语句时，P_5(1.5)的函数值已计算完毕，然后带值返回到引用的表达式，即P_5获得了所计算的值，其余类推。

3.形参与实参的结合

参数的结合是实现程序单位间信息传递的重要途径之一，以后我们还要介绍程序单位间传递信息的其它方法。

（1）实参与形参之间的赋值结合

从函数子程序的结构中可知，形式参数是变量名，而实在参数却可以是常数、表达式和已赋值的变量或下标变量。

当引用函数子程序时，若实在参数是常数或表达式，或内、外部函数引用，则在函数子程序中不能对这样的形式参数再赋值，这种形式的结合被称为值的结合。即把实在参数的值直接赋给与之对应的形式参数后再进行运算。

例4　计算下列函数

$$f(x)=e^{-x^2}$$

当x为0.1，0.2，0.3，0.4时的函数值。

要求用函数子程序结构编写计算程序。源程序如下：

```
      REAL FUNCTION FX(M)
      REAL M
      FX=EXP(-M)
      RETURN
      END
C MAIN PROGRAM IS FXM
      PROGRAM FXM
      X=0.1
      DO 10 I=1,4
        FXE=FX(X*X)
        WRITE(*,100) X,FXE
        X=X+0.1
10    CONTINUE
100   FORMAT(1X,'F(',F4.1,' )=',F10.7)
      END
```

在这个程序中，函数子程序的结构是非常简单的，在函数子程序体中只有一个算术赋值语句。该函数子程序 FX 的

类型为实型，且有一个实型的形式参数M。

当执行主程序时，则对X赋以初值，然后进入循环。这里共循环4次，每循环一次就引用一次函数子程序FX，计算出一个函数值后再为下一次循环作准备。

在引用函数子程序FX时，给出的实在参数是一个实型表达式X＊X，因此，系统将表达式的值计算出来以后才进行形实结合。第一次引用，计算出表达式的值为0.01，此时的函数子程序相当于下列语句：

```
FX=EXP(-0.01)
RETURN
```

计算出FX的值为0.99005，通过函数子程序名将FX的函数值带回到主程序中并输出。用同样的方式可计算出其余的函数值。

（2）实参与形参之间的换名结合

当引用函数子程序时，若实在参数为变量名或下标变量，则不能将实在参数的值直接赋给形式参数，而是用实在参数名去替换相应的形式参数的名字，这种结合方式称为名的结合。名的结合实际上是用实在参数的地址去替换形式参数的地址。对于下标变量，则根据下标表达式的值计算出它在该数组中的位置，然后再用其地址去替换对应的形式参数的地址。

例5　编写一个求

$$f(x, y) = y^2 - \frac{2x}{y^4}$$

的函数子程序及其主程序。

```
FUNCTION FXY(X,Y)              函数子程序
F=Y*Y
```

```
      FXY=F-2.0*X/F**2
      END

      PROGRAM FUNXY                    主程序
      READ(*,'(2F6.2)') A,B
      FN=FXY(A,B)
      WRITE(*,100) A,B,FN
100   FORMAT('A=',F6.2,'  B=',F6.2,
     1'  F(A,B)=',F10.6)
      STOP
      END
```

这里定义的是一个具有2个形式参数的实型函数FXY，其两个形参的类型也是实型。

在主程序中，第一个可执行语句READ按直接格式从键盘上读2个数据分别送A、B中，接着执行算术赋值语句：

FN＝FXY（A，B）

它的右端是引用函数子程序FXY，这里用两个实型变量A和B作为FXY的实在参数。因此，在进行形实参数结合的时候，不能直接将A和B的值赋给形式参数X和Y，而是分别用实型变量A和B去替换形式参数X和Y。然后再执行函数FXY，直到遇到END语句时，则结束FXY函数子程序的执行并由函数子程序名FXY将函数值带回到引用的地方，再赋给实型变量FN，最后输出计算结果。

如果将主程序改写为：

```
PROGRAM FUNXY
DIMENSION A(2)
READ(*,'(2F6.2)')(A(I),I=1,2)
FN=FXY(A(1),A(2))
   ⋮
```

改写的程序中定义了一个两个元素的一维实型数组，对

数组元素的赋值是用READ语句实现的。该数组的第一、二个元素的值分别作为引用函数FXY的第一、二个实在参数。它与形式参数的结合仍然是名的结合，这时的函数子程序相当于下列语句：

```
F=A(2)*A(2)
FXY=F-2.0*A(1)/F**2
```

函数子程序除了由函数名带回函数值外，也可以用实在参数带回所需要的值。

（3）实参和形参是数组名的结合

形式参数也可以为数组，这称作形式数组，它必须由DIMENSION语句或类型语句来定义。与形式数组相结合的实在参数可以是数组名或下标变量。

若形式参数是数组名，则与之对应的实在参数也必须是数组名。在进行形实参数结合时，将从实在参数数组的第一个元素开始，按照排列顺序与形式数组的对应元素相结合。

形式数组与实在数组结合时，形式数组的大小必须小于或等于实在数组的大小。否则将没有与之对应的数组元素。这是因为FORTRAN编译系统在生成可执行程序时，已将常界数组分配了存储单元，而运行时无法进行再分配。

从函数子程序的讨论中已经看出，引用函数子程序是很方便的。但是，引用函数子程序时应注意某些情况的发生，以免产生一些预想不到的问题。

例6　分析下列程序的执行情况。

```
PROGRAM FUNCAL                    (主程序)
REAL I
I=2.0
S=F(I)+I
```

```
      WRITE(*, 100)I, S
100   FORMAT(1X, 'I=, 'F5.0, '␣␣S=', F7.0)
      END
      FUNCTION  F(A)              (函数子程序)
      A=A+1.0
      F=A
      END
```

这个程序中定义了一个实型函数子程序F，它有一个实型形式参数A。

执行上述程序时，主程序FUNCAL中的第一个算术赋值语句对实型变量I赋初值2.0，接着执行算术赋值语句：

S=F(I)+I

该语句的右端表达式中引用了函数子程序F。按表达式的执行顺序应先引用函数子程序F，然后求出表达式的值并赋给实型变量S，从表面上看，似乎S的值应为5.0，但实际上确是6.0。其原因是主程序在引用函数子程序F时，实在参数I与形式参数A以名的方式进行结合，而在函数子程序中形式参数又是一个赋值形式参数，执行语句A=A+1.0实际上就相当于执行I=I+1.0，即得到结果为3.0并赋给I，同时函数F也得到函数值3.0。返回主程序后进行表达式计算，这时的实型变量I的值不是2.0，而是3.0。因此，表达式的值为6.0。

按照表达式的计算顺序，S=I+F(I)是否也应为6.0，但计算机输出的结果为5.0。这与函数优先计算不符。产生这个结果的原因：一是没有优先处理函数引用，而是顺序进行的；二是在计算F(I)时，已将I的值传送到堆栈中保存，返回的函数值与堆栈中I的值相加，即使在函数子程序中改变了I的值，但对堆栈中的值不会改变，所以得到结果为

5.0。

如果把语句S＝F（I）＋I改写为语句

S＝（I＋1.0）＋F（I）或

S＝I＋（1.0＋F(I)）

计算的结果均为6.0，这种情况说明表达式计算满足加法结合律。

若再将语句S＝F（I）＋I改为：

S＝F（I）＋F（I）＋F（I）

则得到S的值为12.0而不是9.0。

通过这个例子说明，在引用函数子程序时，特别是对赋值形式参数的处理要小心，否则将会影响计算的结果。

本例者在其它计算机上运行，其结果如下：

S＝I＋F（I）
S＝F（I）＋I } 结果　S＝6.0

S＝（I＋1.0)＋F（I）
S＝I＋（1.0＋F（I））} 结果　S＝7.0

S＝F（I）＋F（I）＋F（I）} 结果　S＝12.0

这说明系统在处理表达式时优先处理函数引用，而后再处理其它。即计算顺序为：

函数引用→括号→乘方→乘或除→加或减。

7.3　子例程子程序

上节中介绍的函数子程序，它除由函数名带回函数值外，还可以用形实参数结合的方式带回附加的值，不管其参数个数或参数种类如何繁多，计算过程怎样复杂，但它主要的作用还是以返回函数值为主。因此，对独立出来的计算过

程可能得到多个值，或者根本就没有值的情况，函数子程序就不适用了。对于这一类问题，可以用子例程子程序的形式来处理。

7.3.1 子例程子程序的结构

在FORTRAN语言中，几乎任何问题的处理都可以用主程序和子例程子程序（或函数子程序）来实现。

1.子例程子程序的一般结构

子例程子程序是以SUBROUTINE语句开头并以END语句结束的程序单位。它的结构形式是：

SUBROUTINE SUB〔（〔〈形式参数表〉〕）〕

〈说明语句部分〉 ⎫
〈可执行语句部分〉⎭ 子程序体

END

其中：

SUBROUTINE SUB〔（〔〈形式参数表〉〕）〕是子例程子程序语句。

SUB是子例程子程序名。

〈形式参数表〉是该子例程子程序中所需要的形式参数名，形式参数名之间用逗号分隔。根据需要可设置多个形式参数或者没有形式参数。当没有形式参数时，该语句为如下两种形式：

SUBROUTINE SUB（　　）

或　SUBROUTINE SUB

〈说明语句部分〉用于定义形式参数名或该程序单位内的FORTRAN名及其类型。

〈可执行语句部分〉是对子例程子程序的计算过程进行处

理的语句序列。

END是子例程子程序的结束语句。

2.子例程子程序的调用

（1）调用子例程子程序的形式

调用语句CALL是一个可执行语句、它的形式是：

CALL SUB〔（〔〈实在参数表〉〕）〕

其中：

SUB为调用的子例程子程序名，它是由SUBROUTINE语句定义的。

〈实在参数表〉为待选项，它可以是一个以上的实在参数 a_1，a_2，…，a_n。实在参数可以是常数、表达式（包括函数引用）、变量名、下标变量、数组名等。多个实在参数彼此之间必须用逗号“，”隔开。

当子例程子程序为无参子例程子程序时，则可用下面形式的调用语句：

CALL SUB（　　）　　或

CALL SUB

例7　计算多项式

$$P_7(X)=7.8X^7+2.1X^6-5.5X^5+0.23X^3+1.25$$

当X=0.1，0.2，…，1.0时的值。

编写出源程序为：

```
SUBROUTINE PNF(N,A,M,X,PNX)          子程序
INTEGER N,M,I,J
REAL A(8),X(10),PNX(10)
DO 10 I=1,M
   PNX(I)=0.0
   DO 20 J=1,N
      PNX(I)=PNX(I)*X(I)+A(J)
```

```
20        CONTINUE
10      CONTINUE
        RETURN
        END
```

```
        PROGRAM POLYNO                                   主程序
        DIMENSION D(8),X1(10),PN(10)
        WRITE(*,'(''ENTER M=? < M<=10'',
       1'' >,N=? < N<=7 >'')')
        READ(*,'(2I3)') M,N
        WRITE(*,'('' ENTER D(I)=?,I=1...N'')')
        READ(*,'(5F5.2)') (D(I),I=1,N)
        WRITE(*,'(''ENTER X1(I)=?,I=1...M'')')
        READ(*,'(5F4.1)') (X1(I),I=1,M)
        CALL PNF(N,D,M,X1,PN)
        WRITE(*,100) (X1(I),PN(I),I=1,M)
100     FORMAT(1X,'POLY(',F4.1,'  )=',
       1F14.7/1X)
        STOP
        END
```

这个源程序由一个主程序和一个子例程子程序组成。

在子例程子程序PNF中提供了5个形式参数，其中2个整型形参N和M，分别用于传递多项式的次数和自变量个数，其余3个形式参数为实型数组A、X和PNX，分别用于传递多项式系数、自变量值和多项式值。

在主程序中，由CALL PNF（N，D，M，X1，PN）语句调用子程序PNF，并向子程序提供了5个实在参数，其中N（整型）和M（整型）分别为多项式的次数和自变量个数；3个实型数组D、X1和PN，分别用于存储多项式的系数，自变量值和返回多项式之值。

执行调用语句CALL时，则首先进行形式参数与实在参数的结合，即分别用5个实在参数的名字去替换子例程子程序中对应的形式参数名，然后可得到子例程子程序的可执行部分如下：

```
      DO 10 I=1, M
        PN(I)=0.0
        DO 20 J=1, N+1
          PN(I)=PN(I)*X1(I)+D(J)
20      CONTINUE
10    CONTINUE
```

执行形实参数结合后的程序可得到M个N次多项式的值且被存储于PN数组中。当子例程子程序遇到END语句时，则结束子程序的执行并由PN数组带回所计算的多项式的值。

这时再由主程序输出PN的值。

（2）调用语句的执行过程

CALL语句的执行是按下列方式进行的：

计算所有以表达式方式出现的实在参数的值，即对表达式求值。

实在参数与对应的形式参数结合。其结合方式与函数子程序的结合方式相同。

执行指定的程序体。

当执行到RETURN语句或END语句时，控制返回到当前的CALL语句的后继语句处。并由与形式参数对应的实在参数带回运行的结果。

上例主程序中的调用语句CALL执行后，由PN数组带回多项式的值。

```
ENTER  M=? (M<=10), N=? (N<7)
␣10␣␣8↙
```

```
ENTER  D(I)=? , I=1, 2, …, N
7.8  2.1  -5.5  0.0  0.23↙
0.0  0.0  1.25↙
ENTER  X1(I)=? I=1, 2, …, M
 .1   .2   .3   .4   .5↙
 .6   .7   .8   .9  1.↙
POLY(␣0.1␣␣)=␣␣␣␣␣␣1.2501800
POLY(␣0.2␣␣)=␣␣␣␣␣␣1.2503100
              ⋮
```

编写和调用子例程子程序时应注意以下几点：

（1）子例程子程序的符号名只标识该子例程子程序，没有值的意义，所以没有数据类型之分。也不能在子例程子程序中对其赋值。但它是全局名，所以在程序中的任何地方都不能有与之相同的名字。

（2）子例程子程序中的形式参数可以是变量名、数组名。也可以没有形式参数。

（3）实参和形参在个数、顺序、类型方面都必须一一对应。

7.3.2 可调数组与假定大小数组

引用函数子程序和调用子例程子程序时，首先进行实在参数与形式参数的结合。当遇到形式参数是数组名时，则对应的实在参数也必须是数组名或下标变量，并且要求实在数组的大小应大于或等于形式数组的大小。在前面的示例中，形式数组与实在数组都具有相同的大小，所以它们从对应数组的第一个元素开始或从指定的数组元素开始按数组的排列顺序一一对应结合。

除了维数固定的常界数组以外，FORTRAN还提供可调数组。

1.可调数组

在定义数组时，除了用整型常数作维上界之外，还可以用整型变量作维上界，用这种形式定义的数组称为可调数组，其维上界的确定必须在引用该数组之前，而数组是在一个程序单位之前的说明语句中定义的。所以，在主程序中显然不可能定义可调数组，只可能在函数子程序或子例程子程序单位中出现可调数组。

可调数组必须是形式数组，即可调数组名必须是形式参数。其维上界也最好选用整形变量的形式参数或者出现在同一程序单位中的整型公用变量名。

可调数组的引入，有利于通用程序的设计，使存储单元得到充分的利用。

例8　求多项式$P(X)=a_0X^n+a_1X^{n-1}+\cdots+a_{n-1}X+a_n$的值，用可调数组来实现。

```
      SUBROUTINE PNF(N,A,M,X,PNX)
      REAL A(N),X(M),PNX(M)
      DO 10 I=1,M
         PNX(I)=0.0
         DO 20 J=1,N
            PNX(I)=PNX(I)*X(I)+A(J)
20       CONTINUE
10    CONTINUE
      RETURN
      END
```

这个子例程子程序不需要进行任何修改，编写一个主程序，给出实在参数就可求由实在参数所决定的任何次多项式的值。若要计算下列多项式：

$P_3(X)=1.5X^3+2.1X^2+1.3X+1.56$

$P_2(X)=-0.75X^2+2.32X+0.015$

$$P_5(X) = 0.9X^5 + 2.4X^3 - 1.3X^2 + 1.34$$

在X=－0.5，0.5，1.5时的值，其主程序如下：

```
      PROGRAM PN235
      DIMENSION X(3),C(6),D(3)
      WRITE(*,'(''INPUT X(I)=?,I=1...3'')')
      READ(*,'(3F5.1)') (X(I),I=1,3)
      DO 10 I=1,3
        WRITE(*,'(''INPUTN=?<N=2,3,5>'')')
        READ(*,'(I2)') N
        WRITE(*,'(''INPUT C(J)=?,J=1...N+1'')')
        READ(*,50) (C(J),J=1,N+1)
        CALL PNF(N+1,C,3,X,D)
        WRITE(*,100) N,(X(J),D(J),J=1,3)
10    CONTINUE
50    FORMAT(10F6.3)
100   FORMAT(1X,'N=',I2,/,
     1('POLY(',F4.2,'  )=',F10.7/1X))
      STOP
      END
```

2.假定大小数组

可调数组作形式参数，引用函数子程序或子例程子程序时，导致实在数组与形式数组的结合并以实在数组的大小来确定形式数组的大小，给程序设计带来了方便。

除了可调数组以外，还可以把星号“＊”放在数组的最末维上，这样定义的形式数组叫作假定大小数组。

假定大小数组的名字必须是形式参数，它的元素个数无法确定，只有实在参数与形式参数结合后，以实在数组的大小来确定假定大小数组。如：

```
SUBROUTINE SUB(A, B, C, D, M)
DIMENSION A(*), B(10, *)
```

INTEGER C(M, *), D(2, 2, *)

形式数组A是一维的假定数组。B是已知第一维的维上界，第二维是假定的，它是一个二维假定数组。第三个是一个整型形式数组，第一维由整型变量M给出其维上界，第二维是星号“*”，则是不定的。因此，它是一个整型假定数组。第四个是一个整型形式数组，第一、二维用整常数作维上界，第三维不确定，它是三维假定数组。

假定大小数组的前几维若由整常数作维上界，则是常界假定大小的数组。若前几维中出现有整型变量作维上界的，则是可调的假定大小的数组。不管是那一种形式的假定大小的数组，均由引用或调用时的实在数组的大小来确定其大小。

7.4 等价语句与公用语句

7.4.1 等价语句

等价语句EQUIVALENCE是一个非执行语句，它们放在程序单位第一个可执行语句之前。在同一程序单位内，它可用来说明两个以上的变量或数组共享同一个存储单元。

1.等价语句的格式

EQUIVALENCE(〈变量表〉)〔,(〈变量表〉)〕…

其中：

〈变量表〉可以是两个或两个以上的变量名、数组名或数组元素名，相互之间用逗号分隔。

出现在圆括号内的所有元素构成一个区域，称为等价域，同一等价域的变量分配同一个存储单元。如：

EQUIVALENCE(R, S), (A, B, C)　(1)

EQUIVALENCE(A, B, C, D)　(2)

DIMENSION A(100), D(100)　(3)

EQUIVALENCE(A, D)

语句(1)中建立了两个等价域，第一个等价域中的两个变量R和S共用一个实型量存储单元。第二个等价域中有实型变量A、B和C，它们共用一个实型量存储单元。其余类似。

2.EQUIVALENCE语句的规则

(1)函数子程序或子例程子程序的形式参数名不能在等价域中出现。

(2)在同一程序单位中，同一个变量名不能出现在多个等价域中或与同一数组的两个或两个以上的元素等价。

(3)连续的数组元素只能按顺序存放,否则，不允许。

(4)若与公用数组等价，则可使公用数组得到值。

(5)在等价语句中，等价变量的类型之间不能转换。

例9　用等价语句编写的程序示例。

```
      PROGRAM EQUIV
      EQUIVALENCE (A,B)
      EQUIVALENCE (B,C)
      A=10.0
      WRITE(*,10) A,B,C
      B=20.0
      WRITE(*,10) A,B,C
      C=30.0
      WRITE(*,10) A,B,C
10    FORMAT(1X,'A=',F6.0,'  B=',
     *F6.0,'  C=',F6.0)
      END
```

执行这个程序可得到下列数据：

A=␣␣␣10.␣␣B=␣␣␣10.␣␣C=␣␣␣10.

A=　　　20.　　B=　　　20.　　C=　　　20.

A=　　　30.　　A=　　　30.　　C=　　　30.

从程序中的等价语句可知，实型变量A和B共用一个存储单元并构成一个等价域；在后一个等价语句中，实型变量B和C共用一个等价域的存储单元。实型变量B在两个等价域中同时出现，因此，实型变量B将两个等价域连成了一个等价域。即相当于下列等价语句：

EQUIVALENCE（A，B，C）

7.4.2　公用语句

利用公用语句建立公用块是程序单位间传递信息的又一途径，它允许不通过形实参数结合就可以定义和引用相同的数据，并共享有存储单元。在同一个程序单位中，公用语句可以定义无名公用块和有名公用块。

1.公用语句的一般形式

COMMON〔/〔〈公用块名〉〕/〕〈变量表〉〔〔，〕/〔〈公用块名〉〕/〈变量表〉〕…

其中：

COMMON是公用块语句定义符，标志公用块语句的开始。若省去〈公用块名〉，则表示该块为无名公用块；〈变量表〉可以是公用块中的变量名、数组名或数组说明符，相互之间用逗号隔开。

（1）无名公用块的形式

COMMON C_1，C_2，…，Cn　或

COMMON //C_1，C_2，…，Cn

其中：C_1，C_2…，Cn为无名公用块中的公用元素，它可以是变量名、数组名或数组说明符，相互之间用逗号分隔。无名公用块的标志是没有公用块名或是两个连续的斜杠等。如：

```
COMMON A, B, C, D
COMMON //X, Y, Z, F(4)
COMMON I1, I2, NA(10)
```

（2）有名公用块的形式

COMMON /〈公用块名〉/〈公用元素表〉〔/〈公用块名〉〈公用元素表〉〕…

如.

```
COMMON /COM1/A, B, C
COMMON /COM2/A(4), D
COMMON /COM/J, K, L
```

这三个公用语句分别说明了有名公用块COM1、COM2、COM。在COM1中定义了三个实型公用变量A，B和C。在COM2中定义了实型公用数组A，具有四个数组元素，还有一个实型公用变量D。公用块COM中定义了三个整型公用变量J，K和L。它们的单元分配是从各自公用块的第一个存储单元连续地按公用元素出现的先后顺序来分配存储区域。

2.对COMMON语句的限制

（1）公用块中的元素名不允许是形式参数或函数名。

（2）在同一个程序单位中，公用语句可以出现多个，在同一公用块中的所有元素按它们在COMMON语句中出现的顺序分配公用块中的存储单元。

（3）在公用语句中，任一公用变量名、数组名或数组说明符只能出现一次，否则是错误的。

（4）公用块的大小等于公用块中所有元素所要求的总存储字节数。例如：

COMMON A, B, C, D, E(2), I, N

中说明了四个实型公用变量A，B，C，D。两个元素的公用数组E和整型公用变量I、N。它们占用无名公用块的前28个字节的存储区域，即：

A	B	C	D	E(1)	E(2)	I	N
←4字节→	←4字节→	←4字节→	←4字节→	←4字节→	←4字节→	←2字节→	←2字节→

（5）所有程序单位中的无名公用块的大小可以不同，但是，相同名字的有名公用块的大小必须相同。

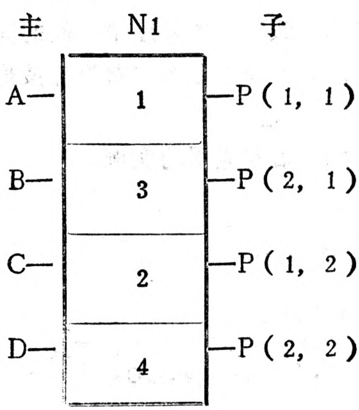
（6）在可执行程序中，具有相同名字的公用块的存储区有相同的第一个存储单元。所有无名公用块的存储区有相同的第一个存储单元，这就导致不同的程序单位间的同名公用块中的对应单元相结合。而不象形参和实参那样按形实参数的个数、类型、位置一一对应。

例10　下面是COMMON语句应用的实例。

```
      PROGRAM MAIN
      COMMON /N1/A,B,C,D
      CALL SUB1
      WRITE(*,10) A,B,C,D
10    FORMAT(1X,'A=',F3.0,'  B=',F3.0,
     *//1X,'C=',F3.0,'  D=',F3.0)
      END
```

```
      SUBROUTINE SUB1
      COMMON /N1/P(2,2)
      WRITE(*,'(''INPUT P(1,1)...P(2,2)'')')
      READ(*,10) ((P(I,J),J=1,2),I=1,2)
      WRITE(*,20) ((P(I,J),I=1,2),J=1,2)
10    FORMAT(4F4.0)
20    FORMAT(1X,'P(1,1)=',F3.0,'  P(2,2)='
     *,F3.0,//1X,'P(2,1)=',F3.0,'  P(2,2)='
     *,F3.0)
      END
```

在这个程序的子例程子程序SUB1中定义了公用块N1，说明了一个二维4个元素的公用数组P。在主程序中 定 义 了一个公用块N1，说明了四个实型公用 变 量 A、B、C和D。同名公用块N1具有相同的第一个存储单元。所以，即 有 下列对应关系：



若输入数据为：

␣1.␣2.␣3.␣4.↙

则输出数据为：

P(1, 1)=␣1.␣␣P(2, 1)=␣3.

P(1, 2)= 2.　P(2, 2)= 4.

A=␣1.␣␣B=␣3.

C=␣2.␣␣D=␣4.

3.公用语句与等价语句联用

等价语句EQUIVALENCE可以和公用元素建立等价关系。但必须注意：

(1)EQUIVALENCE语句不能使用两个不同的公用块中的存储域成为等价，也不能使同一公用块中的元素建立等价关系。如：

```
COMMON /NAME/A, B, C/NA1/E, F, G
EQUIVALENCE(A, E), (A, B)
```

这里定义了两个公用块，块名为NAWE和NA1。在NAWE中说明了三个实型公用变量A，B，C。在NA1中说明了三个实型公用变量E，F，G。

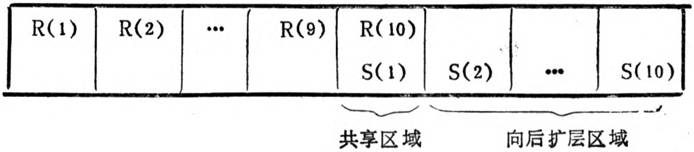
在EQUIVALENCE语句中，建立了两个等价域，第一个等价域要求公用块NAME中的变量A与公用块NA1中的变量E等价。第二个等价域要求公用块NAME中的变量A和B等价。因而等价语句中所建立的两个等价域都是错误的。

(2)可用EQUIVALENCE语句扩展公用块，这种扩展只能向后而不能向前扩展。如：

```
COMMON /ABC/R(10)
REAL S(10)
EQUIVALENCE (R(10), S(1))
```

其中R是公用块中的数组，共有10个数组元素；通过等价语句建立了公用数组R和数组S的等价关系，即是数组元素R(10)和数组元数S(1)等价，因而使得公用区ABC从第

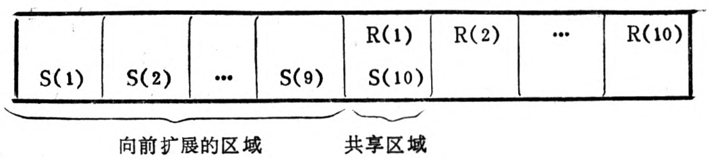
10个元素开始向后扩展。



若改为下列语句：

```
COMMON   /ABC/R(10)
REAL  S(10)
EQUIVALENCE  (R(1), S(10))
```

则使得公用块向前扩展，这是不允许的。



4.SAVE语句

SAVE语句称为保留公用块定义语句，作用是从定义公用域的子程序中返回后保留该块的定义。

在FORTRAN语言标准中，所有的公用块都是静态分配的，都已自动地被保留。因此，SAVE语句在此无多大作用。

习题七

1.设
$$Y(X)=\begin{cases}1+\sqrt{1+X^2}, & 若X<0\\ 0 & 若X=0\\ 1-\sqrt{1+X^2}, & 若X>0\end{cases}$$

从而计算

$$F=K+Y(A+Z)$$

$$G=\frac{Y(A)+Y(C)}{Y(B)}$$

$$H=Y(\cos(X))+Y(\sin(X))$$

其中K，A，B，C，X，Z由输入语句给出，并输出F、G、H之值。

第八章　文件及输入输出系统

本章将讨论FORTRAN语言文件的基本概念、处理以及有关的语句。

8.1　文件的基本概念

1.记录

一个FORTRAN 77文件是由一个或多个记录组成的。记录是字符或数值的序列。有三种类型的记录：格式记录、无格式记录以及文件结束记录。

（1）格式记录由处理系统所能表示的字符序列组成，其长度以字符度量，以回车换行作为一个格式记录的结束。格式记录必须和格式输入输出语句配合使用。

（2）无格式记录是数值序列，不需要系统转换和解释，并且没有记录结束的标志（物理表示）。在不需要编辑和干预时，则可用无格式记录进行存储或恢复信息。

（3）文件结束记录就是由ENDFILE语句输出的，它作为文件的最末一个记录。但是，实际上并没有一个真实的记录与它相对应，这只是文件结束的一个标志。

2.文件

FORTRAN文件分为内部文件和外部文件。内部文件是指一个字符变量，字符数组或字符数组元素，它作为输入输出数据的来源或处理结果而被存储。外部文件是指在外部

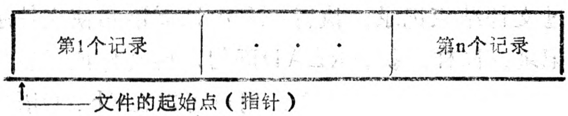
设备上的文件或外部设备本身。

文件具有如下特性：

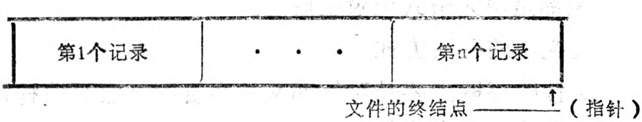
一个文件可以有一个名字，文件名是操作系统能够识别的一个字符串，并与所使用的操作系统的文件取名规则相一致。同一文件名可带有一个扩展名或其它信息。例如：SYS：A·TEXT和#4：A·TEXT等。

一个存储于外部设备上的文件，在读写该文件之前必须执行打开文件的操作。由于每个文件都由若干个记录组成，所以，打开文件后，有一个叫作文件定位指针的指示器指向文件中某个位置。根据文件定位指针指出的位置可对记录进行存取。如果文件定位指针指示的位置不正确，则存取就会出错。有下列几种情况：

（1）一个文件的起始点就是文件的第一个记录前的位置。对文件施行打开操作时，文件定位指针则指向文件的第一个记录之前的位置。如下图所示：

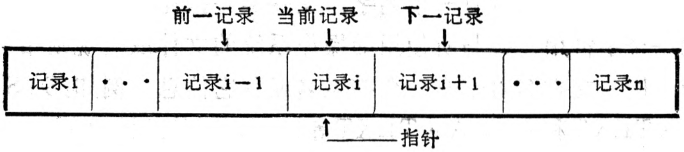


（2）一个文件的终结点是文件的最后一个记录之后的位置。在对文件进行操作时，若文件定位指针已指向文件的终结点，则表示读/写操作已经完成了。可用下图表示：

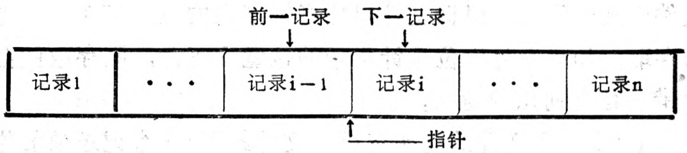


（3）如果文件定位指针指向文件的第i个记录内，则第i个记录被称为当前记录。第i-1个记录称为前一记录，第

i＋1个记录则称为下一个记录。而i∈〔1，n〕。用图表示为：



（4）如果文件定位指针指向两个记录之间，则没有当前记录，而只有前一记录和下一记录。如图所示：



写一个打开的顺序文件时，文件指针在起始位置，文件中的旧数据被废除。顺序写完后，文件指针处于末端位置，但未超过文件结束记录。执行ENDFILE语句使文件越过文件结束记录。同样，执行READ语句，使文件位置也处于末端（没有越过文件结束记录）。用户可以在READ语句中使用“END＝”选择项来检查是否读进一个文件结束记录。

外部文件可作为格式的、无格式的文件打开，内部文件只能是格式的。格式文件全部是由格式记录组成的，无格式文件全部是由无格式记录组成的。

3.文件的存取方式

文件的存取主要是对外部文件而言的，它有两种存取方式，即顺序存取和直接存取。

顺序文件的记录具有一个次序，它由被写入的顺序所决定。这些文件不能用选择项“REC＝”来读或写，因为它将

作为直接文件进行处理。若所写的记录超过该文件原先的结束界限时，系统就扩充顺序存取文件，但这要取决于该物理设备在该文件末尾是否还有存储空间。

直接（即随机）存取文件的顺序是记录编号的顺序。可以按任意次序去读写文件记录。

打开随机文件时，每个记录只有一个编号。这种文件必须建立在外存储器上，不能删除已写过的记录，但可以重写新的记录内容，且只能用随机存取的输入输出语句来处理。

8.2 文件的打开与关闭

要对数据文件进行存取，首先必须用OPEN语句打开数据文件，确定文件的存取方式、格式等。操作完毕还必须关闭所有打开的文件。

1.打开数据文件语句

一个文件的打开是由OPEN语句实现的。用它向系统提供必要的信息，把一个文件的部件号与外部设备或在外部设备上的文件连接起来。

OPEN语句的格式：

OPEN（U，FILE=〈文件名〉〔，STATUS=st〕〔，ACCESS=ac〕〔，FORM=fm〕〔RECL=rl〕）

其中：U为所要求的部件号说明符，它必须作为第一个参数出现，且为一个整形变量或整型常数。U的取值范围：

若U为二字节整型变量时，则取值为：

$$-32768 \leqslant U \leqslant 32767$$

文件名必须为第二个参数。如果指定的文件名为空（即

FILE='␣'），则在程序运行时出现提示信息：

IO ERROR：BAD FILE NAME

S# 8, P# 28, I#146

TYPE 〈SPACE〉 TO CONTINUE

这时敲〈SPACE〉键（即空格），则退出当前运行程序，系统重新初始化。

st为字符表达式，表示文件的状态（即新文件或旧文件）。如果不选择此项，则所打开的文件为OLD状态。St的值可为OLD或NEW。如果值为OLD，则表示所打开的文件已经存在。若值为NEW，则表示所打开的文件为新文件。

ac为字符表达式，它指定文件的存取方式。其值可为SEQUENTIAL（顺序）或DIRECT（直接）。前者为顺序存取方式，后者为直接存取方式。若不选择此项，则隐含为顺序存取方式SEQUENTIAL。

fm为字符表达式，它指定记录的格式，其值可为FORMATTED（有格式）或UNFORMATTED（无格式）之一。如果存取方式是顺序的，则不选择此项时为格式的，即取值为FORMATTED，否则为UNFORMATTED（即为无格式的）。如果为直接存取方式，不选择此项则为UNFORMATTED，否则为格式的，且必须选择FORMATTED。

rl为整型表达式，是以字符为单位的记录长度。此参数用于直接存取文件，并且要求必须有此项。

注意：

（1）待选项彼此间的次序是任意的。

（2）格式中的整型表达式中的变量，必须在此语句之

前赋值。若字符表达式为变量名时，也必须在此语句之前赋值。

（3）若部件号U的值为0，则为键盘或显示器。

例1
```
CHARACTER*10 FNAME
WRITE(*,'(A$)')'OUTPUT FILENAME?'
READ(*,'(BN, A)') FILENAME
OPEN(7, FILE=FNAME, ACCESS= SEDUENTI
AL', *STATUS='NEW')
```

这里打开的文件名由字符变量FNAME的值所决定，与之相连接的部件号为整常数7。存取方式为顺序的，且为一个新文件。

2.关闭数据文件语句

通过OPEN语句打开的文件，操作完毕后必须进行善后处理，解除文件和部件之间的连通关系。这个工作由CLOSE语句来完成。CLOSE语句的形式是：

CLOSE（U〔，STATUS=st〕）

其中：U为整型表达式，与OPEN语句中的说明相同。这里的表达式U指出的部件号，必须是OPEN语句中指定的部件号。

st为字符表达式，取值可为KEEP或DELETE，不选择此项，则为KEEP。若指定为KEEP，则执行CLOSE语句后，该文件仍然存在。若指定为DELETE，则执行CLOSE语句后，部件号U所连接的文件被删去（即不再存在）。

在FORTRAN程序执行中所打开的文件，如果没有明确的用CLOSE语句关闭，则主程序执行结束时系统将自动关闭全部打开的文件。如象CLOSE中选用了KEEP一样，

并且将它们全部保存。如：

```
CLOSE(7, STATUS='DELETE')
CLOSE(NO, STATUS='KEEP')
CLOSE(I1)
```

8.3 其它编辑描述符

在第二章中已经介绍了一些常用的格式编辑描述符（即′、I、F、E、X、H和/），基本上满足一般输入输出格式的处理。这里将再介绍一些格式编辑描述符的用法。

1.字符型量的输入输出

字符型量的输入输出，除用编辑符′和H格式以外，还可用字符编辑描述符实现输入输出。

字符编辑描述符的形式是：

rAw或rA

其中：w是字符个数，r是重复次数。

Aw编辑描述符由字段宽度w决定了字符输入输出表项的长度。对于输入，若指定字段宽度w大于或等于输入表项的长度时，则从输入字段中取最右边的字符，否则以左端对齐，不足部分补空格。对于输出，若指明的字段宽度w大于输出表项的字符数时，则左端输出w－n（输出表项长度）个空格符，否则，输出表项最左端的w个字符。

对于不带字段宽度w的格式符A，则以输入输出表项的长度为准。

例2 字符编辑描述符的应用示例。

```
1  CHARACTER*10 A, B*5
2  WRITE(*, '(1X, A')'INPUT CHARAC-
```

```
      TERS(<10)'
3     READ(*,'(A5)')A
4     WRITE(*,'(A)')A
5     READ(*,'(A)')B
6     WRITE(*,'(A10)')B
            ⋮
```

第1个语句定义了长度分别为10和5的变量A和B。

第2个语句要求用A编辑描述符输出字符常数，这里则按给出的表项长度输出。

第3个语句要求输入5个字符并赋给字符变量A。由于说明语句中给出字符变量A的长度为10，则输入5个字符赋给字符变量A的最左端，后面补5个空格符。如输入字符“ABCDE”，则字符变量A的值为：

$$\underbrace{\text{ABCDE␣␣␣␣␣}}_{10个字符}$$

而接着输出的记录与此相同。

执行第5个语句时，要求按字符变量B的长度输入，若输入字符串为A1234，则执行第6个语句的结果为：

$$\underbrace{\text{␣␣␣␣␣A1234}}_{10个字符}$$

2.比例因子编辑描述符

比例因子编辑描述符的形式是：

kP

其中：k称为比例因子，P为编辑描述符标志字符。

P编辑描述符为F和E编辑描述符指定比例因子，直到遇到下一个P编辑描述符为止。在每个输入输出语句执行开始时，其比例因子设置为零。比例因子按下列方式影响正常的

编辑：

(1)用F和E编辑（若输入字段中无指数）输入以及用F编辑输出时，它使外部表示的数等于内部表示的数乘以10^K。例如：

READ（*，'（F5.2，1PF5.2）'）X，Y

若输入记录：

12·34 12·34↲

则赋给X的值为12.34，而赋给Y的值为123.4。若有输出语句：

WRITE（*，'（1X，F5.2，2X，2PF10.2）'）X，Y

则可得输出记录为：

12·34␣␣␣␣12340·00

F5·2　2X　2PF10.2

（2）用F和E编辑输入时，若输入字段中有指数，则比例因子不起作用。

（3）用E编辑输出时，产生的量的基本实常数部分被乘以10^K并且指数减去K。例如，若内存中A的值为·1256000E␣␣3，则用输出语句：

WRITE（*，'（1X，1PE14·7）'）A

显示的结果为␣1·2560000E+02

1PE14·7

3．$、BN和BZ编辑描述符

（1）BN和BZ编辑描述符

BN和BZ编辑描述符用于解释数值输入域中的空格符。在每一个输入输出语句开始执行时，均置为BZ，它使得除

打头空格外的空格符都作为零处理。如果格式控制程序处理一个BN编辑描述符，则在后面输入域中的空格被掠过，直到处理完一个BN为止，掠过空格的效果是把输入域中所有空格字符好象右对齐一样来处理。即打头的空格数等于掠过的空格数。如：

```
READ(*,'(I4, I5)')IX, IY              (1)
READ(*,'(BZ, I4, I5)')IX,IY           (2)
READ(*,'(BN, I4, I5)')IX,IY           (3)
```

若输入记录为：

␣1␣2 ␣␣␣23↵

则语句（1）和（2）将得到一样的结果，即：

IX=102, IY=23

而语句（3）得到的值为：IX=12，IY=23，即空格不起作用。

（2）美元符号编辑描述符

在设计输入输出编辑格式控制程序时，往往是格式控制程序结束，则导致当前记录的数据传输也随之而结束，如果格式控制程序遇到的最后一个编辑描述符是美元符 $时，则它将自动地禁止记录结束，而允许后面的输入输出语句继续读或写同一记录。它通常被用于键盘输入语句前提示信息的输出，使得用户输入的信息与提示信息位于同一行上。例如：

```
WRITE(*,'(1X, A $)')'INPUT␣FILEN
UMBER='
READ(*,'(BN, I2)')N
```

执行这里的输出语句时，则按指定格式说明在屏幕上显示信息：

INPUT FILENUMBER=

由于输出格式说明的最后一个编辑描述符是 $，所以，使得输出记录未结束，因而后面的输入记录将在输出记录之后的同一行上显示出来。输入文件部件号：2↵则显示信息如下：

INPUT FILENUMBER=␣2↵

8.4 文件的输入输出

上节中所使用的OPEN语句和CLOSE语句，它们只能为文件的操作提供必要的信息和善后处理。要将信息存放到文件中或从文件中读出信息都必须通过I/O语句方可达到目的。下面将分别介绍I/O语句及其应用。

8.4.1 输入输出语句

1.输入语句

输入语句主要用于从输入设备上或外部设备上的文件中输入所需要的数据。

输入语句的形式是：

READ(U〔, fm〕〔, REC=rn〕〔, END=s1〕〔, ERR=s〕)ILIST

其中：U为整型表达式，表示部件号，其值为已打开文件的部件号。它必须作为第一个参数出现。

fm为格式说明符，若为格式读，则必须作为第二个参数出现。其值可分为：

(1) fm为格式语句的标号。

(2) fm为通过赋标号语句ASSIGN赋给格式语句标号

的整型变量。

（3）fm可用直接的格式编辑字符串所替换，其形式为：

'（〈格式编辑字符串〉）'

rn是一个整型表达式，为直接文件存取时使用，其值代表存取的记录号。如果没有REC＝rn选择项，则从文件的当前记录位置顺序向下读。

sl为任选的语句标号，当读到文件结束记录时，则转到指定的语句标号处继续执行。若没有此项选择，则读到文件结束记录时会产生运行错误而终止程序的执行。

s是任选语句标号。当在输入过程中产生错误时，则转移到标号为s的可执行语句继续执行。若无此项，则输入过程中产生错误时会发生运行错误，且终止程序的运行。

ILIST是输入变量表。

2.输出语句

输出语句主要用于将计算机处理好的信息传送到指定的设备上或指定外部设备的文件中。

输出语句的形式是：

WRITE（U〔，fm〕〔，ERR＝s〕〔，REC＝rn〕）OLIST

其中：U为一个整型表达式，且必须作为第一个参数出现。

fm是格式标识符，对格式输出来说，它必须作为第二个参数出现。

s为任选语句标号，当在输出的过程中产生错误时，则转移到指定的语句标号处继续执行。

rn为一个整型表达式，只有使用直接文件时才选择此项，输出就从记录号为rn的位置开始输出。对于直接存取文

件来说，如果省略REC=rn选择项，则从文件的当前记录开始连续向下写。

OLIST是输出变量表。

8.4.2 顺序文件的建立和检索

顺序存取文件又称为SAM文件，它是 SEQUENTIAL ACCESS METHOD的缩写。

顺序存取意味着必须按记录的自然顺序进行存取，即从第一个记录开始，中间不允许跳过任何记录，只能一个一个地依次进行。

按照顺序存取方式打开的文件是自动定位的。当在OPEN语句中指定ACCESS='SEQUENTIAL'或省略此项时，则文件定位指针指向第一个记录的开始处。I/O语句必须首先对第一个记录进行存取，然后第二个，第三个…，依此类推，直到处理完最后一个记录，遇到文件的结束记录时，才能结束对该文件的存取。

例3 建立一个顺序文件

```
      CHARACTER*10 FNAME
      DIMENSION A(100)
      WRITE(*,'(''INPUT UNIT=?'')')
      READ(*,'(I3)') N
      WRITE(*,'(A\)') 'INPUT FILENAME='
      READ(*,'(BN,A)') FNAME
      OPEN(N,FILE=FNAME,STATUS='NEW')、
      DO 10 I=1,100
10    A(I)=I
      WRITE(N,20) (A(I),I=1,100)
20    FORMAT(10F6.1)
      CLOSE(N)
      END
```

执行这个程序时，将分别提示输入部件号和文件名。这时可从键盘上输入部件号和文件名。OPEN语句打开一个顺序存取方式的文件，即ACCESS='SEQUENTIAL'，格式FORM='FORMATTED'，其状态值为“NEW”（这意味着要建立一个新文件）。由循环语句形成记盘的数据并存放于实型数组A中，然后再由写语句的格式决定每个记录为10个数据，数据字段宽度为6个字符，将数组A的元素顺序地写入文件中。当程序运行结束后，可用操作系统的编辑程序显示所建立的数据文件，也可用编辑程序进行编辑。

注：这里所说的显示、编辑数据文件是对格式记录的文件来说的。对于非格式的数据文件不能这样操作。

下面的数据是已建立的数据文件的记录：

```
   1.0   2.0   3.0   4.0   5.0   6.0   7.0   8.0   9.0  10.0
  11.0  12.0  13.0  14.0  15.0  16.0  17.0  18.0  19.0  20.0
  21.0  22.0  23.0  24.0  25.0  26.0  27.0  28.0  29.0  30.0
  31.0  32.0  33.0  34.0  35.0  36.0  37.0  38.0  39.0  40.0
  41.0  42.0  43.0  44.0  45.0  46.0  47.0  48.0  49.0  50.0
  51.0  52.0  53.0  54.0  55.0  56.0  57.0  58.0  59.0  60.0
  61.0  62.0  63.0  64.0  65.0  66.0  67.0  68.0  69.0  70.0
  71.0  72.0  73.0  74.0  75.0  76.0  77.0  78.0  79.0  80.0
  81.0  82.0  83.0  84.0  85.0  86.0  87.0  88.0  89.0  90.0
  91.0  92.0  93.0  94.0  95.0  96.0  97.0  98.0  99.0 100.0
```

要将已经建立的数据文件中的数据读入到内存使用，必须用读文件的操作来实现。对于格式顺序文件来说，用什么格式来建立就只能用什么格式来读。

例4 读出上面数据文件的内容。

```
      CHARACTER*10 CH
      DIMENSION B(100)
      WRITE(*,'(A$)') 'INPUT UNIT='
      READ(*,'(BN,I3)') N
      WRITE(*,'(A$)') 'INPUT FILENAME='
      READ(*,'(BN,A)') CH
      OPEN(N,FILE=CH)
      READ(N,10,END=20) (B(I),I=1,100)
10    FORMAT(10F6.1)
20    WRITE(*,10) (B(I),I=1,100)
      CLOSE(N)
      END
```

如果要修改顺序文件，则必须将它的全部数据读入内存进行修改，然后，将修改后的数据重新存盘，建立一个新文件。成功之后，则可删去原始文件，保留新建的文件。也可一边读记录，判断找出应修改的数据将其修改，然后存入磁盘。也可达到修改的目的。

8.4.3 直接文件的建立和检索

直接文件是由OPEN语句中ACCESS='DIRECT'选择项来决定的。作为直接文件还必须有RECL=rl选择项，由它决定记录的长度，且每个记录的长度相同。这样打开的文件又称为随机存取文件。由于这种文件可按记录号进行任意存取，不涉及记录号的顺序，因此使用起来方便。

例如：一个单位的工资信息、人事档案信息等就是属于这种记录长度相同的信息，因此在处理这些信息时，尤其适合于使用直接文件进行存取。

例5 直接文件的建立。

```
      DIMENSION A(10,10)
      WRITE(*,'(A)') 'UNIT=?',
     *'ERCORD LEN=?(<=10)'
      WRITE(*,'(A)') 'ARRAY BOUND(M,N<=10)'
      READ(*,'(4I3)') NO,LEN,M,N
      WRITE(*,'(''FILENAME'',
     *'' IS DIRDATA.TEXT'')')
      LEN1=LEN*5+1
      OPEN(NO,FILE='DIRDATA.TEXT',ACCESS=
     *'DIRECT',STATUS='NEW',FORM='FORMATTED'
     *,RECL=LEN1)
      DO 20 J=1,M
         DO 10 I=1,N
            A(J,I)=I+10*(J-1)
10       CONTINUE
         WRITE(NO,'(10F5.0)',REC=J)
     *         (A(J,I),I=1,N)
20    CONTINUE
      CLOSE(NO)
      END
```

用这一程序可以建立一个直接格式文件，记录长度由LEN所决定，且LEN≤10。当LEN＝10时，则每个记录可存放10个四字节的实型数据，用记录长度LEN1＝LEN＊5＋1得到记录长度的字节数为51字节（注意：格式记录长度包含一个字节的记录结束符）。由于记录是格式的（即以字符形式写入），所以可以用文本显示命令或文本打印命令将文件的内容显示在屏幕上或打印在宽行纸上，以便检查。

现在用操作系统的编辑程序显示建立的文件内容如下：

1.	2.	3.	4.	5.	6.	7.	8.	9.	10.
11.	12.	13.	14.	15.	16.	17.	18.	19.	20.
21.	22.	23.	24.	25.	26.	27.	28.	29.	30.
31.	32.	33.	34.	35	36.	37.	38.	39.	40.
41.	42.	43.	44.	45.	46.	47.	48.	49.	50.
51.	52.	53.	54.	55	56.	57.	58.	59.	60.
61.	62.	63.	64.	65.	66.	67.	68.	69.	70.
71.	72.	73.	74.	75.	76.	77.	78.	79.	80.
81.	82.	83.	84.	85.	86.	87.	88.	89.	90.
91.	92.	93.	94.	95.	96.	97.	98.	99.	100.

对于上面建立的直接格式数据文件，也可以用一个程序来读文件的内容。

例6 检索直接文件的记录。

```
      DIMENSION A(10,10)
      WRITE(*,'(A)') 'UNIT=?',
     *'ERCORD LEN=?(<=10)'
      WRITE(*,'(A)') 'ARRAY BOUND(M,N<=10)
      READ(*,15) NO,LEN,M,N
      WRITE(*,'(1X,''FILENAME'',
     *'' IS DIRDATA.TEXT'')')
      LEN1=LEN*5+1
      OPEN(NO,FILE='DIRDATA.TEXT',ACCESS=
     *'DIRECT',FORM='FORMATTED',RECL=LEN1)
      DO 10 J=1,M
        READ(NO,20,REC=I) (A(I,J),J=1,N)
10    CONTINUE
15    FORMAT(4I3)
20    FORMAT(10F5.0)
30    OPEN(100,FILE='CONSOLE:')
      WRITE(100,'(1X,10F6.0)')
     *       ((A(I,J),J=1,N),I=1,M)
      END
```

程序中第一个OPEN语句打开的是一直接文件，这里省略了STATUS='OLD'选择项，由于预置值为UNFORMATTED、OLD，所以必须选择FORM='FORMATTED'。于是DIRDATA·TEXT文件就被指定为一个已存在的直接格式文件，它与部件号10相连接。

第二个OPEN语句打开的是设备文件CONSOLE：（即控制台文件名，实际上用星号字符“*”就可以了），也可用打印机文件名PRINTER：代替，从而实现与打印机的连接。语句中省略了“ACCESS=”、“FORM=”和“STATUS=”三个选择项，由OPEN语句预置值可知，CONSOLE：是一个已存在的顺序格式文件。它与部件号100连接在一起。执行WRITE语句时，则将结果显示出来，如下所示：

1.	2.	3.	4.	5.	6.	7.	8.	9.	10.
11.	12.	13.	14.	15.	16.	17.	18.	19.	20.
21.	22.	23.	24.	25.	26.	27.	28.	29.	30.
31.	32.	33.	34.	35.	36.	37.	38.	39.	40.
41.	42.	43.	44.	45.	46.	47.	48.	49.	50.
51.	52.	53.	54.	55.	56.	57.	58.	59.	60.
61.	62.	63.	64.	65.	66.	67.	68.	69.	70.
71.	72.	73.	74.	75.	76.	77.	78.	79.	80.
81.	82.	83.	84.	85.	86.	87.	88.	89.	90.
91.	92.	93.	94.	95.	96.	97.	98.	99.	100.

程序中没有用CLOSE语句关闭DIRDATA·TEXT和CONSOLE：两个文件。程序执行完毕后文件自动关闭，其STATUS='KEEP'。

8.4.4 格式文件的建立和修改

格式文件是由格式记录构成的，它既适合于顺序文件也适合于直接文件。格式文件是由OPEN语句中FORM='FORMATTED'选择项所决定的。若打开的文件为顺序存取方式，则可省略此项。格式文件是一种应用很广泛的数据文件。

例如：我们要建立一个通讯录文件，就可以采用格式记录，为了便于文件的操作，最好选用直接存取方式。

例7　建立一个通讯录文件。

设姓名为NAME，性别为SEX，年龄为YEAR，通讯地址为ADDRES，电话为TEL。其程序如下：

```
      CHARACTER NAME*8, SEX*4, YEAR*4,
     *            ADDRES*30,TEL*16,FN*10
      WRITE(*,'(A$)') 'ENTER FILENAME='
      READ(*,'(BN,A)') FN
      OPEN(10,FILE=FN,ACCESS='DIRECT',
     *  STATUS='NEW',FORM='FORMATTED',
     *  RECL=63)
      I=1
10    WRITE(*,'(A)')'ENTER NAME=?<1-8 CHAR>'
      READ(*,'(A)') NAME
      IF(NAME.EQ.'0') GOTO 50
      WRITE(*,'(A)') 'ENTER SEX=**'
      READ(*,'(A)') SEX
      WRITE(*,'(A)')'ENTEP YEAR=?<1-3 CHAR>
      READ(*,'(A)') YEAR
      WRITE(*,'(A)') 'ENTER ADDRESS=?',
     *                 '<1-30 CHAR>'
```

```
      READ(*,'(A)') ADDRES
      WRITE(*,'(A)') 'ENTER TELEPHONE',
     *               '=?<1-16 CHAR>'
      READ(*,'(A)') TEL
      WRITE(10,20,REC=I) NAME, SEX,
     *               YEAR, ADDRES, TEL
      I=I+1
      GOTO 10
20    FORMAT(A8,A4,A4,A30,A16)
50    CLOSE(10)
      END
```

执行这个程序时，系统将首先提示输入文件名，然后按照程序的执行顺序输入姓名、性别、年龄、通讯地址和电话。若录入的姓名的值为字符“O”，则表示输入信息结束，否则将继续输入以后的各项信息。于是可生成下列记录形式的文件：

××…××	××××	××××	××…××	××…××
NAME	SEX	YEAR	ADDRES	TEL

对于上面所建立的通讯数据文件，可对其中的任何一个记录进行修改。从这一点来说，显然比顺序文件的修改更为方便，只要给出某个记录号，就可将这个记录的全部信息读入内存进行处理。对记录中的字段还可指定要修改的字段编号。

例8　修改通讯录数据文件。

```
      CHARACTER NAME*8, SEX*4, YEAR*4,
     *ADDRES*30,TEL*16,FN*10,COND,COND1
      WRITE(*,'(A$)') 'ENTER FILENAME='
      READ(*,'(BN,A)') FN
      OPEN(20,FILE=FN,ACCESS='DIRECT',
     *        FORM='FORMATTED',RECL=63)
30    WRITE(*,'(A$)') 'DO YOU EDIT'
     *   ,'COMMUNICATION FILE ?<Y/N>'
      READ(*,'(BN,A)') COND
      IF(COND.EQ.'Y') THEN
        WRITE(*,'(A)') 'ENTER THE ',
     *  'NUMBER OF RECORD='
        READ(*,'(BN,I3)') I
        READ(20,40,REC=I) NAME, SEX,
     *                  YEAR, ADDRES,TEL
        WRITE(*,80)NAME,SEX,YEAR,ADDRES,TEL
60      WRITE(*,'(A$)') 'ENTER THE',
     *              ' NUMBER OF TERM<1-5>='
        READ(*,'(BN,I2)') N
        GOTO (1,2,3,4,5) N
        GOTO 70
1       WRITE(*,'(A,A)') 'NAME= ',NAME
        READ(*,'(A)') NAME
        GOTO 50
2       WRITE(*,'(A,A)') 'SEX=  ',SEX
        READ(*,'(A)') SEX
        GOTO 50
3       WRITE(*,'(A,A)') 'YEAR= ',YEAR
        READ(*,'(A)') YEAR
        GOTO 50
4       WRITE(*,'(A,A)') 'ADDRESS= ',ADDRES
        READ(*,'(A)') ADDRES
        GOTO 50
5       WRITE(*,'(A,A)') 'TELEPHONE= ',TEL
        READ(*,'(A)') TEL
50      WRITE(*,'(A$)') 'DO YOU ',
     *               'CONTINUE...?<Y/N>'
```

```
      READ(*,'(BN,A)') COND1
      IF(COND1.EQ.'Y') GOTO 60
      WRITE(20,40,REC=I) NAME,
     *  SEX, YEAR, ADDRES, TEL
      GOTO 30
40    FORMAT(A8,A4,A4,A30,A16)
    ENDIF
70  CLOSE(20)
80  FORMAT('1. ',A,'2. ',A,'3. ',
   *        A,'4. ',A,'5. ',A)
    STOP
    END
```

这个程序被执行时，首先将要修改的记录读入，然后对指定的项进行修改。

8.4.5 无格式文件的建立和检索

无格式文件是由无格式记录组成的。

打开无格式文件是在OPEN语句中选择 FORM='UNFORMATTED'来实现的。只要按这种方式打开的文件，在使用输入输出语句时就不能选择fm项。这时的输入输出语句为：

READ(U〔, REC=rn〕〔, END=sl〕〔, ERR=s〕) ILIST

WRITE(U〔, REC=rn〕〔, ERR=S〕)OLIST

无格式文件的记录是二进制格式的，用选择项FORM='UNFORMATTED'即按二进制格式进行存取。也就是说，系统是严格按照计算机内部的二进制位进行存取的。如果在写文件时用的是无格式，则读入信息也必须按照无格式进行操作。

无格式文件对顺序存取方式或直接存取方式都是适用的。在OPEN语句中，一是选择ACCESS＝'SEQUENTIAL'，或者省略该项。而另一种则是选择ACCESS＝'DIRECT'。前者表示顺序存取方式，而后者表示直接存取方式。当为直接存取方式时，省略"FORM＝"这项，则表示为无格式记录的文件。

对于无格式文件只能在磁盘或磁带上进行存取，不能在顺序设备上进行操作。如：显示器、打印机都不能直接显示或打印无格式文件，因为它们都是顺序格式的设备。

在存取无格式文件时，由于是按机内格式存取，所以不需要进行任何转换。对于实型变量按四字节存取；对于整型变量则按二字节存取；而对于一个字符则以一字节为单位进行存取。

我们以下列两个程序为例，说明无格式文件的用法。

例9　无格式文件的建立。

```
      DIMENSION A(10,10)
      OPEN(15,FILE='FN',ACCESS='DIRECT',
     *STATUS='NEW',FORM='UNFORMATTED'
     *,RECL=40)
      DO 20 I=1,10
         DO 10 J=1,10
            A(I,J)=10*(I-1)+J
 10      CONTINUE
         WRITE(15,REC=I) (A(I,J),J=1,10)
 20   CONTINUE
      END
```

执行这个程序将建立一个直接文件，以无格式记录形式打开，并且以二进制记录为存取单位，没有记录结束标志，同样也没有文件结束标志。

我们从前面已经知道，直接文件的存取以块为单位进行。在UCSD PASCAL操作系统下，外存的一块可以存放512个字节的信息。对于磁盘来说，正好是相邻的两个扇区，每次与磁盘进行信息存取，总是512个字符。所以，直接存取文件必须建立在块设备上。

下面是以无格式记录形式读直接文件，记录长度同存入时相同，并且将文件的数据从打印机上输出。

例10　无格式文件的输出。

```
      DIMENSION A(10,10),B(10,10)
      OPEN(10,FILE='FN',ACCESS='DIRECT',
     *STATUS='OLD',FORM='UNFORMATTED',
     *RECL=40)
      DO 100 I=1,10
         READ(10,REC=I) (A(I,J),J=1,10)
100   CONTINUE
      OPEN(30,FILE='PRINTER:')
      WRITE(30,'(A)') 'UNFORMATTED RECORDS'
      WRITE(30,10) ((A(I,J),J=1,10),I=1,10)
10    FORMAT(1X,10F6.1)
      END
```

执行时，从打印机上输出结果如下：

```
UNFORMATTED RECORDS
  1.0   2.0   3.0   4.0   5.0   6.0   7.0   8.0   9.0  10.0
 11.0  12.0  13.0  14.0  15.0  16.0  17.0  18.0  19.0  20.0
 21.0  22.0  23.0  24.0  25.0  26.0  27.0  28.0  29.0  30.0
 31.0  32.0  33.0  34.0  35.0  36.0  37.0  38.0  39.0  40.0
 41.0  42.0  43.0  44.0  45.0  46.0  47.0  48.0  49.0  50.0
 51.0  52.0  53.0  54.0  55.0  56.0  57.0  58.0  59.0  60.0
 61.0  62.0  63.0  64.0  65.0  66.0  67.0  68.0  69.0  70.0
 71.0  72.0  73.0  74.0  75.0  76.0  77.0  78.0  79.0  80.0
 81.0  82.0  83.0  84.0  85.0  86.0  87.0  88.0  89.0  90.0
 91.0  92.0  93.0  94.0  95.0  96.0  97.0  98.0  99.0 100.0
```

8.4.6 外部文件和内部文件

外部文件就是指存储于外部存储介质上的文件和外部设备本身，和操作系统中所使用的外部命令是同一个意思。那里所指的外部命令是一些可装入内存中执行的程序，而这里所指的外部文件，包括那些由用户生成的程序（可执行的或不可执行的）和可顺序或直接存取的数据文件。又可将这些存于外部存储器上的数据文件称为外部数据文件。它们都用OPEN、READ和WRITE等语句对文件进行操作，且都以记录为单位进行存取。

本章前几节建立的数据文件都属于外部文件。

内部文件是内存中的一个字符变量，数组、字符数组元素。它只有一个记录，其长度与字符变量或字符数组元素的长度相同。内部文件只能是格式化的顺序存取文件。

使用内部文件，可将读入的字符变量的字符转换为数字的、逻辑的或字符的值：将写出的字符变量的字符值转换为它的字符形式。

例如：字符变量A＝'1.234－86.45'，读出为1.234、－

86.45并分别存放到实型变量X、Y中，将X和Y相加送入实型变量Z中，再将实型变量Z的值用写内部文件的方式写入字符变量B中。其程序与执行时输出的结果如下：

```
C EXAMPLE OF INTERNAL FILE I/O.
      CHARACTER*11 A,B
      A='1.234-86.45'
      READ(A,2) X,Y
2     FORMAT(F5.5,F6.2)
      Z=X+Y
      WRITE(B,'(F6.2)') Z
      WRITE(*,'(1X,A)') B
      WRITE(*,'(F18.6)') Z
      END
```

```
-85.22                    （字符变量B的值）
        -85.216000        （实型变量Z的值）
```

8.5 文件定位语句

在FORTRAN语言中，有三个文件定位语句，它们是BACKSPACE语句、ENDFILE语句和REWIND语句。

文件定位语句用于顺序存取文件的定位操作。下面将分别介绍这三个语句。

1.BACKSPACE语句

BACKSPACE语句使连接到指定部件的文件被定位于前一记录的开始处。

BACKSPACE语句的形式为：

BACKSPACE 〈UNIT－SPEC〉

其中：〈UNIT－SPEC〉为所要求的外部数据文件的部件识别符。

如果没有前一个记录，则文件的位置不改变。如果前一记录是文件结束记录，则文件被定位于文件结束记录之前。如果文件的位置在记录的中间，则 BACKSPACE 语句重新定位于该记录的开始。如果文件是二进制文件，则 BACKSPACE语句重新定位于前一字节。

2.ENDFILE语句

ENDFILE语句对指定部件所连接的文件写一个文件结束记录，作为该文件的下一个记录。

ENDFILE语句的形式是：

ENDFILE 〈UNIT－SPEC〉

其中：〈UNIT－SPEC〉为所要求的外部数据文件 的部件识别符。

执行ENDFILE语句将定位指针指向文件结束记录之后的位置。如果还要对该文件进行操作，则必须执行 BACKSPECE或REWIND语句后才能对该文件进行数据的存取。

注意：在直接存取文件上进行ENDFILE操作时，将使写在新文件结束以外位置上的全部记录丢失。

3.REWIND语句

REWIND语句使文件的定位指针指向指 定 文件的起始位置。

REWIND语句的格式为：

REWIND 〈UNIT－SPEC〉

其中：〈UNIT－SPEC〉为所要求的外部数据文件的 部件识别符。

以上所描述的三个文件定位语句，除 了 用 于 顺序文件外，也可根据需要用于直接文件的存取。但对于直接文件的存取一般可用记录号，除非在存取直接文件记录时不用记录

号，才能用定位语句来处理。

在定位语句中，如果部件识别符是一个整型变量，则必须用圆括号“(”和“)”括起来。

例11　利用定位语句定位记录。

```
      REAL A(5,5),B(5,5)
      DO 10 I=1,5
         DO 10 J=1,5
            A(I,J)=5+I+J-1
10    CONTINUE
      OPEN(2,FILE='FN',ACCESS='SEQUENTIAL',
     *STATUS='NEW')
      WRITE(2,30) ((A(I,J),J=1,5),I=1,5)
      ENDFILE 2
      REWIND 2
      READ(2,'(5F4.0)',END=20)
     *((B(I,J),J=1,5),I=1,5)
20    WRITE(*,40) ((B(I,J),J=1,5),I=1,5)
      CLOSE(2)
30    FORMAT(5F4.0)
40    FORMAT(1X,5F5.0)
      END
```

执行这个程序时，建立一个格式顺序文件，共存放5个记录，每个记录5个数据段。当记录存完后，在文件的末尾写一个文件结束记录。这时，该文件共有六个记录，文件定位指针指向文件结束记录之后。后面的READ语句则是从该文件中读出数据，放在数组B中。因此，必须将文件的定位指针指向该文件的起始点才能读到正确的数据。于是必须在READ语句之前执行REWIND语句。输出的数据如下：

```
6.  7.  8.  9.  10.
7.  8.  9.  10. 11.
```

8. 9. 10. 11. 12.

9. 10. 11. 12. 13.

10. 11. 12. 13. 14.

如果回退一个记录，则可用BACKSPACE语句，上例中可将REWIND语句删去，在此处插入3个BACKSPACE 2语句，就可将文件定位指针回退到第四个记录之首。这时读出的记录则是最后两个记录的数据。

第二部分　PASCAL 语言程序设计

本部分介绍中华学习机CEC—Ⅰ上PASCAL语言的基本命令以及纯量、子界、集合、数组、串、记录、文件和指针等类型的数据结构及其在程序设计中的应用方法。

第九章　PASCAL语言基础

9.1　语言的基本元素

1.基本符号

PASCAL由字母、数字和特殊符号这三类基本字符组成。

字母：A～Z，a～z

数字：0～9

特殊符号：+、-、*、/、∧、=、<、>、(、)、[、]、.、;、:、,、#、$

赋值运算符：　:=

关系运算符：=、< >、<=、>=、<、>和IN

子界分隔符：..

注释：(*和*)

2.保留字

保留字是系统保留的具有固定意义的单独符号，它不能再作为用户自定义的标识符。例如：PROGRAM、TYPE、VAR、LABEL等。

3.标识符

标识符是用来表示程序、过程、函数、类型、常量和变量等名称的符号。

标识符必须以字母开头，其后可跟字母或数字。例如：

合法的标识符	不合法的标识符	
CUSTOMER	MAX TEMP	（出现空格）
A	X−1	（出现减号）
A12B9	2AB	（以数字开始）
BCDEFG	VAR	（用了保留字）

4.分隔符

PASCAL把空白符、行结束符和注释看作分隔符。

5.运算符

PASCAL提供如下四种运算符，并按顺序指出了它们的优先级：

①乘除运算符：*、/、DIV、MOD

②加减运算符：+、−

③逻辑运算符：NOT、AND、OR

④关系运算符：=、< >、<、>、<= >=和 IN

同一优先级的运算符，从左到右计算。括号内的表达式优先并独立于前后运算符。

如果乘法和加法的两个操作数是整型，那么结果也是整型；如果其中一个或两个是实型，则结果为实型。

由运算符、标识符等就可以构成表达式。

9.2 程序的结构

PASCAL程序的结构如下：

```
PROGRAM 〈程序名〉(〈程序参数表〉)；
    〈说明部分〉
BEGIN
    〈语句序列〉
END.
```

其中：程序名就是给PASCAL程序取的名字；程序参数表表示程序与外界的联系，一般是一个或多个文件变量名。程序通过这些参数调用外部文件。最常用的程序参数是INPUT和OUTPUT，它表示该程序有输入和输出操作。

说明部分包含五段：标号说明部分

常量说明部分

类型定义

变量定义

过程和函数

每一段要么出现一次或根本就不出现，且必须严格遵守上述顺序。

1.标号说明部分

标号又称语句标号，它表示程序转移去执行的那个语句的标志。语句标号由标号名和冒号组成。在PASCAL语言中，凡是程序执行部分使用的语句标号，都必须在标号说明部分进行说明。标号说明的格式是：

LABEL 〈语句标号表〉；

语句标号表由一个或多个语句标号组成，其中标号限制为一至四位无符号整数。例如：

```
LABEL 10, 301, 9999;
IF A>0 THEN GOTO 10;
   ⋮
10:   B:=C+100;
```

注意：标号之间用逗号（，）隔开，最后用分号（；）结束。

2.常量说明部分

一个命了名的常数值称为常量。保留字CONST用于常

量值命名。例如：

```
CONST MAX=1024;
        MIN=-MAX;
  PASSWORD='OK';
```

常量可以是串、字符或数值。

3.类型定义部分

程序中的每个变量都有一个与其相联系的类型。类型既决定了该变量所能取的值，也决定了对该变量所能执行的运算。PASCAL提供了四种标准类型，且允许用户通过类型定义部分来定义自己所需变量的类型。保留字TYPE用于类型定义，其格式如下：

TYPE 〈标识符〉=〈类型〉;

　　　　······

例如：

```
TYPE  STR1=STRING[18];
      LETTER='A'..'Z';
      DAY=(MON,TUE,WED,THU,FRI,SAT,SUN);
```

4.变量说明部分

变量是在程序执行过程中可以改变其值的量，使用前必须在变量说明部分加以说明，才能在程序中引用。保留字VAR用于变量说明，其格式如下：

VAR 〈标识符表〉：类型；

　　　　······

标识符表由一个或多个标识符组成，标识符之间用逗号（，）隔开，最后用分号（；）结束。

例如：

```
VAR  NUM:INTEGER;
     REDIUS,RESULT:REAL;
     WORKDAY,HOLIDAY:DAY;
     STR:STR1;
     CH:LETTER;
```

5.过程和函数说明部分

过程说明是用于定义题目中某个子任务的程序。其一般形式如下：

PROCEDURE 〈过程名〉(〈参数表〉)；

〈程序体〉

程序体是一个或多个语句，它表示程序需要完成的一系列操作。

函数说明用来定义一个计算过程并返回计算的值。请参见第十三章。

9.3 标准纯量类型

前面已提及，变量有四种标准纯量类型，这就是：整型、实型、字符型和布尔型。

1.整型

PASCAL中的整型数取值范围从－32768到32767，每个整型数在内存中占两个字节。

在程序中，整型变量只能用类型标识符INTEGER来说明。例如：

VAR X, Y：INTEGER；

整型常数必须在常量部分说明。例如：

```
CONST  PAGESIZE=60;
       BIGGEST=1000;
       SMALLEST=-BIGGEST;
```

2.实型

实型数的值域为－3.40282E38至3.40282 E38。实数值精度是32位二进制数字，接近7位十进制数字。

PASCAL的实数用两种方法表示：十进位表示法和科学表示法。

十进位表示法就是小数形式的表示方法。它是人们日常生活中习惯使用的方法。例如：

3.5，0.0，＋5.01，29.99

－3.50，1.0，－115.134，－29.999

科学表示法就是指数形式的表示方法，它与计算机的浮点表示是一致的。例如：

＋5.02E＋2，－5.0E＋2，7.92E9

＋5.02E－2，－3.8E－2，－82.1E2

实型变量只能用类型标识符REAL来说明。例如：

VAR X2，Y：REAL；

实型常数在常量部分说明。例如：

CONST PI＝3.1415926535；

3.布尔型

布尔变量在变量说明中被说明。例如：

VAR P，Q：BOOLEAN；

布尔变量只能取TRUE或FALSE值，并约定FALSE＜TRUE。一个布尔变量在内存中占两个字节。

布尔类型主要用于程序中的流程控制和逻辑判断。

布尔常量也必须在常量说明部分说明。例如：

CONST SWICH＝TRUE；

4.字符型

字符型变量在变量说明中被说明。例如：

VAR A, CH : CHAR;

一个字符变量占内存一个字节。

字符常量在常量说明部分说明。例如：

```
CONST  ENDCHAR='$';
       BEGINCHAR='P';
```

注意：字符型常量和变量的数据只能是一个字符，不能是一串字符，且用单引号括起来。例如：'8'、'?'、'A'、'='，'␣'等都是字符。

9.4 PASCAL的通用语句

语句是PASCAL语言的执行部分，每条语句就是程序动作的一部分。PASCAL语言的通用语句按其结构可分成两大类：基本语句和构造型语句。

9.4.1 基本语句

基本语句结构简单，它包括赋值语句，转移语句，空语句和过程语句。

1.赋值语句

赋值语句是最简单的语句。它的功能是确定变量的内容。其格式为：

〈变量名〉：=〈表达式〉；

作用：将表达式的值传送到变量名中。

其中变量名已在变量说明时确定；：=是赋值运算符，它表示将运算符右端的表达式的运算结果送到运算符左端变

量所在的存储单元；表达式允许为常量、变量、函数、算术表达式或布尔表达式。

例1 计算函数

$f(x)=\frac{\pi}{2}+3x^2$ 在x=3.1处的值。

```
PROGRAM CALFUN(OUTPUT);
CONST PI=3.1416;
VAR X, F: REAL;
BEGIN
X:=3.1;
F:=PI/2+3*X∧2;
WRITE(X, F)      (输出结果)
END.
```

例2 已知两数分别存放在两个单元中，现要求把此两数对调位置。

```
PROGRAM SWAP;
VAR A, B, C: REAL;
BEGIN
  A:=3.2;
  B:=8.3; (A, B单元分别存放已知数)
  C:=A;
  A:=B;
  B:=C
END.
```

2.转移语句

转移语句用于改变程序流程。

格式：GOTO 〈标号〉

作用：无条件转到标号所指定的语句去继续执行。

例3 键盘输入一系列数据，统计输入正数的个数，当

输入负数或零时，程序停止。

```
PROGRAM  INNUM(INPUT, OUTPUT);
LABEL  111, 222;
VAR  C:INTEGER;
     R:REAL;
BEGIN
  C:=0; (*$G+*)(编译选择项)
111:READ(R); (键盘输入数据)
  IF R<=0 THEN GOTO 222; (输入的数小于等于零,
                                转输出)
  C:=C+1;
  GOTO 111;
222:WRITE('C=', C); (输出结果)
    WRITE('EXIT')        (输出停止信息)
END.
```

注意：使用GOTO语句前，必需在使用标号的那个程序段的标号说明部分说明标号。

标号的作用域就是说明它的那个程序段。因此不能从外部转进或转出一个过程或函数。

PASCAL编译选择项对GOTO语句的默认值为G-，即不允许用户在程序中使用GOTO语句，若使用了GOTO语句则产生一个编译语法错。因此，用户如果要在程序中使用GOTO语句，必须在程序中加上编译选择项(* $G+*)，才允许使用GOTO语句。

3.空语句

空语句是没有任何符号的语句，当程序需要一条不产生任何动作的语句时，即可使用它。

例4 下面的程序先显示一个“中”字，然后等待用户

敲入任何一键，再往下显示新的内容。

```
PROGRAM SPACESTATDEMO(INPUT,OUTPUT);
BEGIN
  WRITELN('     *     ');
  WRITELN('     *     ');
  WRITELN('*********');
  WRITELN('*    *    *');
  WRITELN('*********');
  WRITELN('     *     ');
  WRITELN('     *     ');
  WRITELN('     *     ');
  WRITELN('ENTER ANY KEY TO CONTINUE')
  REPEAT UNTIL KEYPRESSED;
  WRITE('GOOD BYE! ')
END.
```

其中KEYPRESSED为布尔类型的标准函数。当用户敲任何键时，KEYPRESSED为真，否则为假。

循环语句REPEAT和UNTIL（参见循环语句部分）之间没有任何动作，即相当于有一条空语句，在用户敲入任何一键之前，程序一直处于等待状态，这样便于用户看清前面显示的一幕图文。用户不想再看时，敲入任何一键，程序继续运行。这种方法在编程中经常用到。

4.过程语句

过程语句用于调用用户在说明部分定义的过程或预定义标准过程。过程语句由过程标识符、可选参量表组成。参量表列出了由逗号分隔开的变量或表达式，并用圆括号括起来。程序执行期间遇到过程语句，控制便转到该过程名，并且将可能的参数值传递给该过程。当过程执行完成后，从调

用过程语句的后继语句开始继续执行。过程语句的格式为：

过程名（〈参量表〉）；

例5　求S＝1＋2＋3＋……＋N。

```
PROGRAM PROCDEMO(INPUT, OUTPUT);
VAR M, S:INTEGER;
PROCEDURE CULSUM(N:INTEGER);
VAR I:INTEGER;
  BEGIN
      S:=0;
  FOR I:=1 TO N DO
      S:=S+I;
      WRITELN(S)
  END;
BEGIN          (主程序开始)
  READ(M);
  CULSUM(M)
END.
```

上面程序中READ（M）和CULSUM（M）为过程语句，其中READ为预定义标准过程。

9.4.2 构造型语句

构造型语句是由几个成份组成的一种语句结构。这些语句是：复合语句，条件语句和循环语句。

1.复合语句

复合语句是用BEGIN和END将一系列的语句括起来，从而成为一个整体。复合语句的一般格式是：

BEGIN

　　〈语句序列〉

```
END;
```

其中，语句序列可以是一个或多个语句。

例6　给定半径R＝110厘米的圆铁板一块。若以其圆心为中心，逐次以5，10，15，…，100，105厘米为半径对圆铁板进行圆切割，分别求所切出的各圆环（第一个是圆）的面积。

各圆环的面积为：

$\pi R^2-\pi(R-5)^2=\pi(R^2-(R-5)^2)=\pi(10R-25)$。

```
PROGRAM AREACAL(OUTPUT);
CONST  PI=3.14159;
VAR    R:INTEGER;
       AREA:REAL;
BEGIN
   R:=5;
   WHILE R<=110 DO
   BEGIN
      AREA:=PI*(10*R-25);
      WRITELN(R-5, R, AREA);
      R:=R+5
   END
END.
```

注意：相邻语句彼此之间必须用分号（；）隔开。

2.循环语句

如果需要重复执行某些操作时，可以用PASCAL的循环语句。如果能够预先知道或计算出重复次数，那么用FOR循环，否则用WHILE或REPEAT循环。

（1）REPEAT语句

格式：REPEAT

〈语句序列〉

UNTIL〈条件〉

作用：反复执行指定的语句序列，直到条件满足才退出循环。

这里的语句序列是需要循环执行的部分，又称循环体。这种循环语句的特点是先执行，后检查循环条件。控制重复的条件实际是一个布尔型表达式。REPEAT 循环至少执行一次，因为是先执行语句序列后判断条件。

例7　整数求和直至某数为零时停止。

```
PROGRAM REPEATDEMO(INPUT,OUTPUT);
VAR SUM,NUMBER:INTEGER;
 BEGIN
   SUM:=0;
   REPEAT
   READ(NUMBER);
   SUM:=SUM+NUMBER
   UNTIL  NUMBER=0;
   WRITELN('THE TOTAL IS',SUM)
 END.
```

注意：如果循环体由多个语句组成，那么各语句彼此间必须用分号（；）隔开。循环体中通常要能够改变条件表达式的值，使之朝着满足条件的方向发展，否则可能造成永无休止的循环，称为死循环。

（2）WHILE语句

格式：WHILE〈条件〉DO

〈语句〉

作用：当条件为真时，反复执行指定的语句，直到条件为假时为止。这种循环语句的特点是：先检查条件，然后确定是否循环。如果条件一开始就不成立，则循环一次也不执

行。

例8　当$A>=B^N$时，求最大的N。

```
PROGRAM WHILEDEMO(INPUT,OUTPUT);
VAR A,B,PRODUCT:REAL;
   POWER  :INTEGER;
BEGIN
  READ(A,B);
  POWER:=0; PRODUCT:=B;
  WHILE  PRODUCT<=A DO
   BEGIN
    POWER:=POWER+1;
    PRODUCT:=PRODUCT*B
   END;
  WRITE('LARGEST POWER OF',B,'<',A,
             'IS',POWER)
END.
```

注意：循环体中只能是一个语句，如果是多个语句，必须用BEGIN和END括起来，使之成为一个复合语句。

（3）FOR语句

格式：FOR 〈变量名〉：＝〈表达式1〉TO/DOWNTO 〈表达式2〉DO　　〈语句〉

作用：给变量赋初值（表达式1），执行一次语句；然后增加（TO）1或减少（DOWNTO）1再执行语句，如此重复，直到变量值超过终限值（表达式2）为止。

其中：变量名又称为循环控制变量，表达式1为初值，表达式2为终限值。TO/DOWNTO表示可用TO或DOWNTO。如果是递增1，就用TO；如果是递减1就用DOWNTO。

例9　求N个数的平均值。

```
PROGRAM FORDEMO(INPUT,OUTPUT);
VAR I,N:INTEGER;
   AVERAGE,SUM,NUMBER:REAL;
```

```
BEGIN
  READ(N);SUM:=0;
  FOR I:=1 TO N DO
   BEGIN
     READ(NUMBER);
     SUM:=SUM+NUMBER
   END;
  AVERAGE:=SUM/N;
  WRITELN('AVERAGE IS',AVERAGE)
END.
```

注意：循环体若是多个语句，则每个循环体都必须分别用BEGIN开头，并以END结尾。

例10 某工厂1985年的产值为200万元，计划年增长率为5%，计算并输出1990年至2000年的产值。

```
PROGRAM FORDEMO2(OUTPUT);
VAR VALUE, RATE:REAL;
     YEAR:INTEGER;
BEGIN
VALUE:=200.00;
YEAR:=1985;
RATE:=0.05;
WRITELN('1985 YEAR:200.00');
FOR YEAR:=1986 TO 2000 DO
  BEGIN
     VALUE:=VALUE*(1+RATE);
     IF YEAR>=1990 THEN
     WRITELN(YEAR,'YEAR:',VALUE)
  END
END.
```

3.条件语句

条件语句有两种结构：

（1）IF语句

格式：IF〈条件〉THEN〈语句1〉〔ELSE〈语句2〉〕

作用：当条件成立时执行语句1，否则执行语句2。语句1或语句2执行结束后，顺序执行后继语句。

其中：ELSE根据需要而选用，称为待选项，我们用〔 〕表示。

例11 IF语句示例。

```
IF A<>0 THEN
  IF A>0 THEN
   P:=Q
  ELSE P:=R;
```

表示A为零时没有动作，A为正时把Q值赋给P；A为负时把R值赋给P。为避免产生语法多义性，ELSE子句属于不带ELSE的最近的那个IF语句。因此上面的语句也可以写成：

```
IF A<>0 THEN
  BEGIN
    IF A>0 THEN
     P:=Q
    ELSE P:=R
  END;
```

例12 统计正数和负数的个数。

```
PROGRAM IFDEMO(INPUT,OUTPUT);
VAR POSCOUNT,NEGCOUNT:INTEGER;
    NUMBER:REAL;
BEGIN
  POSCOUNT:=0;
  NEGCOUNT:=0;
  REPEAT
   READ(NUMBER);
   IF NUMBER>=0 THEN
   POSCOUNT:=POSCOUNT+1
```

```
  ELSE  NEGCOUNT:=NEGCOUNT+1
 UNTIL NUMBER=0;(*最后一个是零*)
 WRITELN('THERE WERE', POSCOUNT,
                    'POSITIVE NOS');
 WRITELN('AND',NEGCOUNT,'NEGATIVE NOS')
END.
```

注意：IF和ELSE的对应关系。

（2）CASE语句

格式：CASE〈表达式〉OF

情况标号1：〈语句1〉；

情况标号2：〈语句2〉；

⋮

〔ELSE

情况标号N：〈语句N〉〕

END；

作用：根据表达式的值，分别选择对应的情况标号所示的语句来执行，然后转到END的后继语句去执行。如果没有与表达式的值相符合的情况标号，就转到该语句的下一个语句去执行。

例13 摸拟APPLE PASCAL主菜单及流向控制。

```
PROGRAM PROGRAMFRAM(INPUT, OUTPUT);
VAR CH:CHAR;
PROCEDURE COMPILER;
  BEGIN
    WRITELN('HERE PUT  THE  PROCEDURE
     "COMPILE"');
  END;
PROCEDURE EDITOR;
  BEGIN
    WRITELN('HERE PUT  THE  PROCEDURE
```

```
    "EDIT" ');
  END;
PROCEDURE RUN;
  BEGIN
   WRITELN('HERE PUT  THE  PROCEDURE
    "RUN" ');
  END;
PROCEDURE FILER;
  BFGIN
   WRITELN('HERE PUT  THE  PROCEDURE
    "FILER" ');
   END;
PROCEDURE LINK;
  BEGIN
  WRITELN ('HERE  PUT  THE  PROCEDURE
   "LINK" ');
  END;
BEGIN       (主程序)
  REPEAT
      READ(CH);
      CASE CH OF
       'C ':COMPILER;
       'R ':RUN;
       'E ':EDITOR;
       'L ':LINK
       'F ':FILER;
      END
  UNTIL CH= 'H '
END.
```

主程序中用REPEAT循环语句中嵌入CASE语句，作为主程序框架，这是写大程序的常用方法之一。

以该程序为基础，略加修改，可以较快地搭起一个含多个过程的程序框架，易为初学者模仿。当过程EDITOR单独地调试通过后，再用它来取代程序中的过程EDITOR，这就是PASCAL语句的自底向上的开发方针。

例14　计算函数

$$F=\begin{cases} 0 & \text{当}x=0 \\ (1-x)^{8}+1 & \text{当}x=1 \\ x^{8}+1 & \text{当}x=2 \\ x & \text{当}x\text{为其它任何整数} \end{cases}$$

```
PROGRAM CASEDEMO(INPUT, OUTPUT);
VAR X, F:INTEGER;
BEGIN
  READ(X);
  CASE X OF
  0:BEGIN
       F:=0;
       WRITE('X=0', 'F=', F)
       END;
  1:BEGIN
       F:=(1-X)∧3+1;
       WRITE('X=1', 'F=', F)
       END;
  2:BEGIN
     F:=X∧3+1;
     WRITE('X=2', F=', F)
     END;
  ELSE
```

```
  BEGIN
          F := x;
          WRITE ('x=', x, 'F=', F)
      END
  END
END.
```

习 题 九

1.PASCAL语言的程序结构由几部分组成？它与你熟悉的其它算法语言的结构有什么不同？

2.下列字符串哪些是合法的标识符？

ABCDE, 2ABC, VAR, VARI, TYPE, A␣BC-D,

SA21, SA$, Y12, DE#, 'AB ', 123

3.下列数据哪些是整数？哪些是实数？哪些超过了APPLE PASCAL系统允许的范围？哪些是非法的？

123, 3.46, 20E+4, −4.0E38, 0

2.1E75, 0.0, −370, 35000, −35111

71.50, −2.1E−20, 100.0, 5E5, 0.00001

4.下列符号哪些是字符？

'ABC ', 'P ', '52 ', C, '。', '] '

3, '3 ', 'THIS IS A PEN ', '$ '

5.下列表达式中，哪些结果为真，哪些为假？

5>8, 123<=123, 2=5−2, −3=−3.0

5=5.0, 'A '> 'P ', 'C '>= 'A ', 'A '= 'A '

6.写出对应于下面公式的PASCAL赋值语句。假定所有变量都是实数，并且定义需要的常量。

三角形的边长为a、b和c，其面积为

$$A=\sqrt{S(S-a)(S-b)(S-c)}$$

其中S=(a+b+c)/2

7.写出下列各题的PASCAL语句：

（1）根据变量PENCE是否等于零，打印“ZERO”和“NONZERO”。

（2）已知收入为I元，计算应付所得税T元。设400元以下不收税，400至600元收税率30%其余为40%。此外，当总收入超过4000元时，税率增至50%。

8.鸡兔同笼，共计有M个头，N只脚。编制计算鸡兔各有多少的程序。其中M和N由键盘输入。

9.编写程序计算所有小于500的质数的个数。质数是一个正整数，它不能被1以外的任何整数所整除。检查的一种方法是对每个数确定它是否能被小于它的所有正整数整除。

第十章 PASCAL程序的运行

如果读者希望尽快上机进行实际操作，那么就请依次阅读本章。

10.1 PASCAL系统的启动

PASCAL系统允许用一个或多个驱动器工作。由于中华学习机一般情况下只有一个驱动器（通过扩展槽可增加一个驱动器），用单驱动器和双驱动器启动系统稍有不同，下面分别叙述（请参见第三册《操作系统》中有关UCSD PASCAL的叙述）。

1.单驱动器启动

单驱动器启动分二步：

第一步冷启动。先打开屏幕电源，将APPLE1：系统盘插入驱动器，然后打开计算机电源，等待片刻，屏幕显示下列信息：

```
WELCOME APPLE1 TO APPLE II PASCAL 1.1
BASED ON UCSD PASCAL SYSTEM II.1
CURRENT DATE IS 26-MAY-82
 [C]APPLE COMPUTER INC.1979,1980
 [C]U.C REGENTE 1979
```

再等片刻，屏幕出现命令级提示行：

COMMAND：E（DIT，R（UN，F（ILE，C（OMP，L（INK 这表明冷启动成功。如果只想编辑程序和运行已编

译连接过的代码程序，可从命令级提示行选择相应的E和X（或R）命令，就能工作。但如果还需要编译新程序，那必须进行第二步。

第二步热启动。取出APPLE1：系统盘，插进APPLE0：系统盘，按CTRL—2键，稍等片刻，屏幕显示信息："WELCOME APPLE0"，几秒钟后，屏幕再度出现命令级提示行，这时从命令级提示行可以选择任意一个命令。

2.双驱动器启动

先开屏幕电源，把APPLE1：系统盘插入驱动器（卷号#4:），而把APPLE2：系统盘插入扩展驱动器（卷号#5:），然后打开计算机电源，等待片刻，屏幕上出现"WELCOME APPLE1"信息，再等片刻，屏幕出现命令级提示行，这意味着系统正常运行。用户可以选择需要的命令。

10.2 磁盘的格式化和复制

为了安全地保存原盘，在使用PASCAL之前应复制系统盘。但新盘不能直接复制，必须格式化后，方能在系统内正常工作。

1.盘片的格式化方法

当系统启动之后，取出系统盘，将系统盘APPLE3：插入驱动器#4：中（双驱动器插入#5：中）。由于格式化程序存放在APPLE3：里，这一步不可少。然后键入X命令，屏幕提示和回答如下（回答内容用下横线标出）：

```
EXECUTE WHAT FILE?
APPLE3:FORMATTER ↵ （↵表示回车键）
APPLE DISK FORMATTER PROGRAM
FORMAT WHICH DISK (4,5,9..12)?
```

取出APPLE3盘，并将空盘插入#4　中。敲入：

4↵

NOW FORMATTING DISKETTE IN DRIVE 4

这表示#4：中的磁盘正在进行格式化处理，大约30秒后，格式化结束，系统自动给该盘附上卷号BLANK：。

FORMAT WHICH DISK（4，5，9··12）？

4↵　　（表示还要格式化另一张盘片）

若只敲入↵，则表示格式化工作结束，回到命令级。

如果在格式化过程中，偶尔由于疏忽大意，将系统盘或工作盘放入#4：中，并敲了↵键，例如将APPLE0：盘当成空盘，系统将显示出警告性信息：

DESTROY DIRECTORY OF APPLE0：？

系统盘当然要保护，立即键入N，盘内信息不受破坏。

2.复制系统盘

在命令级键入F，系统进入文件级命令提示行：

FILER：G（ET，S（AVE，W（HAT，N(EW，L(DIR，
　　　R(EM，C(HNG，T(RANS，D（ATE，Q（UIT
　　　或者

FILER：G，S，N，L，R，C，T，D，Q

键入T命令，TRANSFER？

对于这个问题的回答单驱动器和双驱动器稍有不同，假设被复制的源盘是APPLE3：，目标盘是卷号BLANK：（因为格式化后的新盘卷号都是BLANK：）。

（1）单驱动器

取出驱动器中的系统盘，插入APPLE3：盘。以下是回答和显示的信息：

APPLE3:, BLANK:

TRANSFER 280 BLOCK?

Y↵

DESTROY BLANK:?

Y↵

PUT IN BLANK:

TYPE ⟨SP⟩ TO CONTINUE

现在取出APPLE3:盘，插入目标盘，敲空格键（屏幕上用⟨SP⟩表示空格，输入程序或数据时，我们用␣或空白表示空格），几秒钟后，文件程序告诉：

PUT IN APPLE3:

TYPE ⟨SP⟩ TO CONTINUE

如此交换盘的过程重复20次，最后复制完。显示：

APPLE3:→BLANK:

这表明目标盘上已有了APPLE3:盘上的全部信息，包括卷号也复为APPLE3:，不再是BLANK:了。

（2）双驱动器。

将APPLE3:盘放入#4:中，BLANK:盘放入#5:中。回答和显示的内容如下：

#4:, #5:

TRANSFER 280 BLOCK?

Y↵

DESTROY BLANK:?

Y↵

APPLE3:→BLANK:

整个复制过程结束，中间无须交换盘。一块盘复完，系统还处于文件级，如要从文件级返回到命令级，必须先将APPLE0:（或APPLE1:）插入#4:驱动器，再键入

Q，方可达到目的（由于文件管理程序不在系统内）。

10.3 运行PASCAL程序

10.3.1 PASCAL程序的输入

在命令级键入E命令，系统调入编辑程序，显示：

```
>EDIT
   NO WORKFILE IS PRESENT·FILE? (<RET>
   FOR NO FILE <ESC--RET> TO EXIT )
```

其中>EDIT表示系统进入编辑状态。由于系统中没有工作文件，因此编辑程序询问用户是否要工作文件。同时按下ESC键和↵键，则退出编辑状态，返回命令级；而敲↵键，表示没有现成的文件可用作工作文件，用户希望生成一个新的PASCAL程序。屏幕显示编辑级的提示：

```
>EDIT: A(DJST, C(PY, D(LETE, F(IND, I(NSERT,
       J(MP, R(PLACE, Q(UIT, X(CHNG, Z(AP
```

键入I命令，显示插入命令提示：

```
>INSERT: TEXT [<BS>A CHAR, <DEL>A LINE]
         [<ETX>ACCEPS, <ESC>ESCAPES]
```

这时用户便可以输入PASCAL源程序了。

下面以具有简单输入/输出功能的完整程序为例，说明输入PASCAL程序的方法。

例1 源程序的输入练习。

```
PROGRAM EX1A(INPUT,OUTPUT);
 VAR  A,B,C:INTEGER;
 BEGIN
     READ(A,B);
```

```
            C:=A+B;
            WRITELN(C)
        END.
```

若输入过程中敲错了字符，可以用左箭头"◁"键消去一个字符，用CTRL-X删除一行字符。然后再输入正确字符。如果敲一下ESC键，将消去全部输入的字符。当输完上述程序以后，应敲CTRL-C键，使已经输入的程序的字符固定下来，无论再敲左箭头、CTRL-X还是ESC键都不会消去程序的字符；同时使系统退出插入方式返回编辑级命令提示行。

至此已将PASCAL源程序全部输入到内存。在编辑级命令提示行下选Q命令，转到退出命令提示行：

```
>QUIT:
 U(PDATE THE WORKFILE AND LEAVE
 E(XIT  WITHOUT UPDATING
 R(ETURN TO THE EDITER WITHOUT UPDATING
 W(RITE TO A FILE NAME AND RETURN
```

选U命令，系统将输入的上述程序作为工作文件，存入自举盘（即放在#4：中的APPLE0：或者APPLE1：盘）中，并加上文件名SYSTEM·WRK·TEXT。在存盘的过程中系统还会显示出：

```
>QUIT:
 WRITING……
 YOUR FILE IS 100 BYTES LONG.
```

然后，系统返回命令级提示行，再键入R，就能运行上面的程序了。

注意：在键入R之前，检查一下APPLE2：系统盘是否在#5：中；单驱动器APPLE0：在#4：中。

10.3.2 修改程序

由于已经敲过CTRL-C键，输入的程序便被"固定"了，就不能用"◁"或CTRL-X键来修改程序。这时应采用在编辑状态下修改的方法。假定上述程序的第二行有一些错误的输入并且没有及时发现，如：

```
PROGRAM EXIA (INPUT, OUTPUT);
VAR A, B:INTNEGOR;
```

显然，B之后漏掉了"，C"，多出一个字母"N"以及后面应该是"E"而敲为"O"了。

修改出错程序的步骤是：

1.进入编辑状态

如果键入Q、U或R之前发现程序输入字符有错误，系统正处在编辑状态，可立即修改之。如果键入Q、U或R命令之后由语法检查才发现程序有错误，此时系统已处于命令级提示行，要能改错，必须键入E命令，使之转到编辑级状态。

2.移动光标到出错的字符上

利用如下的光标控制键，可以方便地移动光标到指定的字符位置。

◁　光标左移一个字符。

▷　光标右移一个字符。

CTRL-O　光标上移一行。

CTRL-L　光标下移一行。

将光标移到第二行的"："位置上。

3.插入遗漏字符

键入I，进入>INSERT级。键入遗漏字符"，C"（此

时“；”以后的字符立即全部从屏幕上消失，不必惊慌，这些信息并没有真的丢掉）。

敲CTRL-C（原来已消失的字符又全部重现在屏幕上，字符“，C”也加上了），显示：

```
>EDIT：A(DJST……
PROGRAM EXIA（INPUT，OUTPUT）；
VAR A，B，C：INTNEGOR；
  ⋮
```

4.删除错误字符

移动光标到字母“N”上。

键入D命令，系统进入删除命令提示：

>DELETE：〈 〉〈MOVING COMMANDS〉［〈ETX〉 TO DELETE，〈ESC〉TO ABORT］

敲一下“▷”键，字母N立即消失（光标落在字母E上，如果继续敲“▷”键，可消去后面的所有字符；如果发现消去了不应该删除的字符，再敲“◁”键，去掉的字符又会出现在屏幕上）。

再键入CTRL-C，才真正删除了字母N，系统返回编辑级命令提示行，显示：

```
>EDIT：A（DJST……
PROGRAM  EXIA（INPUT，OUTPUT）；
VAR A，B，C：INTEGOR；
  ⋮
```

5.更换字符

移动光标到字母“O”上；

键入X命令，进入相应的更换命令提示行。敲一下E键，字母“O”立即消失而改为E（如果继续敲字符可将“R；”

更换掉，但这里的“R；”是正确的不应该更换。敲“◁”键，换掉的字符又会恢复而出现在屏幕上）。

再键入CTRL-C，才真正更改字符，此时系统返回编辑级，显示：

```
>EDIT : A ( DJST……
PROGRAM  EXIA (INPUT, OUTPUT);
VAR A, B, C: INTEGER;
  ⋮
```

至此，程序中的错误全部改正。

6.转到退出（QUIT）状态

键入Q命令，使系统转到退出命令提示：

```
>QUIT :
 U(PDATE  THE  WORKFILE  AND  LEAVE
 E(XIT  WITHOUT  UPDATING
 R(ETURN TO THE EDITER  WITHOUT UPDATING
 W(RITE  TO  A FILE NAME AND RETURN
```

选U命令，更新工作文件，使修改后的程序存入自举盘，并附上文件名SYSTEM·WRK·TEXT。系统返回命令级提示行。

10.3.3 编译和运行程序

在命令级提示行下选R命令，就调用编译程序，对工作文件进行编译，屏幕显示编译时的情况：

```
PASCAL COMPILER II.1[B2B]
<0>...
EXIA [2091 WORDS]
<3>...
6 LINES
SMALLEST AVAILABLE SPACE=2091 WORDS
```

屏幕显示的EXIA标识符是程序和其过程的名字。〈 〉内的数字是编译到的当前行号，后面跟随省略号，其每一个点表示源程序已通过编译的一行。方括弧[ ]内的数字表示在编译过程中，在该行上最少可用空间的大小。

编译完成后，接着自动运行，屏幕显示：

RUNNING…

5␣4↵（输入数据，用空格␣分隔数据）

9　　（输出的结果）

敲↵键，屏幕又显示出命令级提示行，并将代码工作文件SYSTEM·WRK·CODE存入自举盘。

如果输入的源程序有错误，我们假设程序EXIA少写了变量A的说明。编译程序会发现错误，并显示出错信息：

C：=A<<<<<

LINE5，ERROR104：〈SP〉（CONTINUE），〈ESC〉（T-
　ERTINATE）　E（DIT

这时停止编译，系统等待一个敲键命令。

敲ESC键，退出编译程序，返回命令级。

敲空格键，继续编译，以便发现更多错误。

敲E键，退出编译程序，进入编辑程序。

屏幕顶部显示出错信息UNDECLARED IDENTIFIER，敲一下空格键，出错信息消失，显示编辑级的提示行，按照前面所述的修改程序的方法，可对源程序进行修改。

如果用户的自举盘（APPLE0：或者APPLE1：）存放的文件太多，当键入Q、U时，工作文件不能存入自举盘，无法编译和运行。这时只有将自举盘中的命名文件（用户文件，不是系统文件）清除一些，让出磁盘空间，才能存入。

10.4 工作文件的更新

由于系统每次只允许一个工作文件，因此要想输入新的程序，必须对原有工作文件进行处理。

1.将工作文件作为命名文件存入磁盘

在命令级提示行下，键入F命令，进入文件级命令提示行：

FILER：G，S，N，L，R，C，T，D，Q〔1.1〕

键入S命令，显示和回答如下：

SAVE AS？

EXAMPLEIA↙

自举盘中的SYSTEM· WRK· TEXT和SYSTEM WRK·CODE工作文件就会以名EXAMPLEIA·TEXT和EXAMPLEIA·CODE存入自举盘中。如果要想将命名文件存入用户的目标盘，可用文件级的T命令，将自举盘上的文件EXAMPLEIA·TEXT和EXAMPLEIA·CODE复制到目标盘。

如果有两个驱动器，取出驱动器#5：里的APPLE2：盘，插进已格式化的磁盘，作为存放源程序的目标盘。然后回答：

#5：EXAMPLEIA↙

则自举盘的工作文件的两个版本以名EXAMPLEIA·TEXT和EXAMPLEIA·CODE存入目标盘。

2.清除工作文件

当工作文件已成为命名文件存盘之后，这个工作文件就无继续保留的必要了，应清除掉，让出工作文件区供新的

PASCAL程序使用。清除工作文件的方法是：

在文件级提示行下，键入N命令，显示和回答如下：

THROW AWAY CURRENT WORKFILE?

Y↵

则清掉内存里的工作文件，同时删去自举盘中的SYSTEM·WRK·TEXT和SYSTEM·WRK·CODE两个工作文件。

如回答N，则工作文件仍被保留下来。

3.创建新的工作文件

在命令级键入E，进入编辑级，由于工作文件已被清除，显示：

›EDIT

NO WORKFILE IS PRESENT FILE? (‹RET› FOR NO FILE ‹ESC-RET› TO EXIT)

可用两种方法建立新的工作文件。第一种方法是敲↵键，以新输入的PASCAL程序作为工作文件，这在前面已介绍过，不再赘述。第二种方法是将自举盘上的命名文件作为工作文件。假设命名文件是TREE·TEXT，键入：

TREE·TEXT↵

系统在自举盘上检索文件TREE.TEXT，找到这个文件后，在屏幕上立即显示出文件的内容，编辑程序将TREE·TEXT作为工作文件的文本，并自动在自举盘上建立新的文件SYSTEM·WRK·TEXT。这时工作文件名虽与前次的工作文件名相同，但文件的内容却是新的。如果命名文件TREE·TEXT不在自举盘，可通过文件级的T命令，将它从目标盘传送到自举盘。

如有两个驱动器，则目标盘插入驱动器#5：内，然后

键入：

#5：TREE·TEXT↵

编辑程序也将TREE·TEXT作为工作文件。

习题十

1.单驱动器和双驱动器启动系统的主要不同在哪儿？亲自在你的中华学习机上实践一下系统的启动。

2.为什么要格式化新磁盘？格式化新盘的文件在哪张系统盘上？

3.单驱动器和双驱动器复制系统盘有哪些主要差别？

4.叙述输入一个PASCAL源程序的大致过程。输入过程发现错误（未敲CTRL-C之前）怎么修改程序？敲了CTRL-C后又怎样修改程序？

5.编译和运行程序用什么命令？编译出错怎样处理？

6.在你的中华学习机上编译和运行一个PASCAL程序。

7.为什么要更新工作文件？怎样创建一个新的工作文件？

第十一章　纯量、子界、集合、数组和串类型

前面讨论了PASCAL的标准类型：INTEGER、REAL、BOOLEAN和CHAR类型。这里再讨论PASCAL中要用到的纯量、子界、集合、数组和串类型。

11.1　纯量类型

1.纯量类型定义

纯量类型的说明比较简单，它是该类型的变量可能具有的值的一个表。例如：

TYPE UNITS=(INCHES, FEET,FURLONGS, MILES)；

指出类型UNITS的变量只可能取指定的四个值之一，而不能取别的值。类型UNITS和标准类型一样在变量说明中使用。例如：

VAR SCALL, UNITS;

纯量定义的例子如下：

```
TYPE
 DAY=(MONDAY,TUESDAY,WEDNESDAY,THURSDAY,
      FRIDAY,SATURDAY,SUNDAY;
 RELATIONSHIP=(PARENT,SIBLING,OFFSPRING,
               COUSIN);
 VAR HOLIDAY,WORKDAY:DAY;
     RELATIVE:RELATIONSHIP;
```

2.纯量类型的常量

列在纯量类型说明的值表中的名字是该类型的常量。因此可将纯量常量赋给该类型的变量。例如：

HOLIDAY:=SUNDAY;

RELATIVE:=PARENT;

3.纯量类型的关系运算

关系运算符=、＜＞、＜、＞、＜=、＞=可用于纯量型变量，表达式的结果是一个布尔值。纯量类型的次序是由类型说明中它们出现的先后顺序确定的。因此下述表达式为真：

INCHES＜FEET＜FURLONG＜MILES

标准类型BOOLEAN是一个纯量类型，它由以下说明隐含定义：

TYPE BOOLEAN=(FALSE, TRUE);

4.纯量函数

标准函数PRED（求纯量的前导值）和SUCC（求纯量的后继值）是为纯量型参数定义的。回送的值和参数的类型相同，是参数在定义表中的前导值和后继值。例如：

PRED(FEET)=INCHES（INCHES在FEET之前）

SUCC(FURLONGS)=MILES（MILES在FURLONGS之后）

表的第一个值没有前导值，最后一个值没有后继值。因此求SUCC(MILES)和PRED(INCHES)将要出错。

标准函数ORD（求纯量的次序序号）有一个纯量型参数并回送一个整数值，它是纯量值在其定义表中的次序号码。表中的第一个值，次序数为0，因此ORD(INCHES)

=0；而ORD(FEET)=1(次序号码从0开始计数)。

例1　一个教师从星期一到星期日的工作安排。

星期一、三：讲计算机原理课

星期二　　：学习政治

星期四、五：讲程序设计课

星期六　　：学习英语

星期日　　：自由地安排工作

```
PROGRAM SCALARSDEMO(OUTPUT);
TYPE
DAYS=(SUN, MON, TUE, WED, THU, FRI,SAT);
VAR
    I:INTEGER;
    WORK:DAYS;
BEGIN
WRITE('LOCAL ORDER: ');
WORK:=MON;
WHILE WORK<=SAT DO
      BEGIN
        I:=ORD(WORK);
        WORK:=SUCC(WORK);
        WRITE(I:3) (变量I占3位)
      END;
WRITELN;
WRITELN('WORK  CASE:');
FOR WORK:=SUN TO SAT  DO
      CASE  WORK  OF
  MON:WRITELN('1……TEACH COMPUTER! ');
  TUE :WRITELN('2……STUDY POLITICS! ');
  WED:WRITELN('3……TEACH COMPUTER! ');
```

```
THU : WRITELN ('4……TEACH PROGRAMING! ');
FRI : WRITELN ('5……TEACH PROGRAMING! ');
SAT : WRITELN ('6……STUDY ENGLISH! ');
ELSE WRITELN('0……OPEN! ')
END
END.
```

在这个程序中，纯量类型变量WORK作为循环语句WHILE的控制变量，在变量WORK不大于SAT时，继续不断地循环，输出MON至SAT的序数值。

纯量类型变量WORK作为情况语句的选择表达式，情况标号如MON是纯量类型数据。

11.2 子界类型

1.子界类型定义

一个子界类型可由两个常量定义。常量的类型可以是整数、字符或纯量类型。不允许实数的子界。例如：

```
TYPE  INDEX=1..20;
      LETTER='A'..'Z';
      WEEKDAY=MONDAY..FRIDAY;
```

类型WEEKDAY是纯量类型DAY（称作相关纯量类型）的子界。INDEX的相关纯量类型是整型，LETTER的相关纯量类型是字符型。

子界的定义简单规定了子界的最小和最大值。第一个常量规定了下界，第二个常量规定了上界，下界一定不能大于上界。

2.子界型变量

子界型变量在变量说明部分说明。例如：

```
TYPE  LETTER='A'..'Z';
      DAYS=(SUN, MON, TUE, WED, THU,
            FRI, SAT);
      WEEKDAY=MON..SAT;
VAR   WORKDAY=WEEKDAY;
      FIRSTCHAR, LASTCHAR:LETTER;
```

子界类型的定义和子界变量的说明可以进行合并。例如：

```
VAR  WORKDAY:MON..SAT;
     FIRSTCHAR, LASTCHAR:'A'..'Z';
```

过程READ（读）和WRITE（写）可用于整数和字符型的子界。

例2 求出某月属于哪个季节。

```
PROGRAM  SUBRANGESDEMO(INPUT, OUTPUT);
VAR
  MONTH:1..12;
BEGIN
  READ(MONTH);
  CASE MONTH OF
  2..4:WRITELN(MONTH, 'MONTH IS SPRING.');
  5..7:WRITELN(MONTH,'MONTH IS SUMMER.');
  8..10:WRITELN(MONTH, 'MONTH IS AUTUM-
        N.');
  11..12, 1: WRITELN(MONTH, 'MONTH IS WIN-
             TER.');
  END
END.
```

11.3 长整数类型

PASCAL允许用户自定义长整数类型数。据定义的方法是在保留字INTEGR后带上长度说明。例如：

TYPE BIGNUM=INTEGER〔12〕；

VAR FATS：INTEGER〔25〕；

BIGNUM定义为一长整数类型，属于这种类型的数据能取任何不超过12个十进制数字的整数。第二句说明变量FATS也为长整数型，它可以取小于25个十进制数字的任何整数。

在PASCAL中，一般整数取值范围是－32768到32767，而长整数类型的长度说明可以是不超过36的任何无符号整常数，即长整数类型最大可以取到36位十进制数字。

长整型常量也是允许的。任何其值超过一般整数范围的整型常数自动处理为一长整型常数。

算术运算符＋、－、＊、DIV能用于长整型数据，但MOD却不能。当一个长整型数据赋给一个长整型变量时，如果值的十进制数字多于变量定义的长度则发生溢出。在整型算术运算中，如果中间或最终结果多于36个十进制数字也发生溢出。

整型值可以赋给一个长整型变量，且自动将它转换成相应的长度。但是，却不能将一个长整数值赋给一个整数变量。如果一个表达式中同时包含整型值与长整型值，则整型值被自动转换成长整型值，运算结果为长整型值。长整型和实型是不相容的，它们不能同时出现于一个表达式中，也不能彼此赋值。

所有标准的关系运算符均可以用于长整型数据或长整型与整型相混合的情形。

内部过程STR（LONG，STRG）中的参数LONG可以是长整型值，该过程将LONG的值，转换成一个十进制数字串，送给串变量STRG。内部函数TRUNC亦能接受长整型值作为参数，如果该长整型绝对值小于或等于MAXINT（＝32767为系统定义的整型常量标识符），则TRUNC回送相应的整型值。STR和TRUNC是唯一允许接受长整型值作为参数的内部过程和内部函数。

企图在一个参数表中说明某参数为长整数类型将导致语法错误，但这种限制可以通过定义一个长整数类型来消除。例如：

```
  PROCEDURE BIGNUMBER(BANKACCT: INTEGER
〔18〕);
```

将导致语法错，而

```
  TYPE LONG=INTEGER〔18〕;
  PROCEDURE BIGNUMBER(BANKACCT: LONG);
```

是合法的。例如：

```
  VAR I: INTEGER;
      L: INTEGER〔N〕;（N为<=36的整常数）
      K: REAL;
  BEGIN
    I:=L;（语法错）
    I:=TRUNC(L);（正确）
    L:=-L;（正确，假定L是不多于36位的十进制数字，负号不
               计在数字内）
    L:=I;（正确）
    L:=K;（不接受）
```

K：=L；（不接受）

分配给长整数类型的内存单元总是整数个字。一个INTEGER〔N〕类型的变量占用（N+3）/4+1个字。因此，一个长整数的值的实际限度可以超过说明中规定的十进制数的位数。例如，一个5到8位长的长整型占用3个字，能存放的最大数为99 999 999（占两个字）。

例3 计算中国、美国和印度人口的总数。1976年中国人口数为825,000,000；美国人口数为212,000,000；印度人口数为586,000,000。

```
PROGRAM  LENINTDEMO(OUTPUT);
VAR  CHINA, USA, INDIA: INTEGER [9];
      TOTAL: INTEGER [12];
BEGIN
  CHINA:=825000000;
  USA:=212000000;
  INDIA:=586000000;
  TOTAL:=CHINA+USA+INDIA;
  WRITE(TOTAL)
END.
```

11.4 集合类型

1.集合及其操作

集合理论是数学的一个分支，它是由康托在十九世纪后期创建的。集合论的基础在于集合是由它的成员或元素来定义的。例如：

（1）0至511的所有整数构成一个集合；

（2）西文字母构成一个集合；

（3）西文字符的并又构成另一个集合。

两个集合相等，当且仅当它们的每个元素相同（这里没有涉及元素的顺序）。例如：

集合〔1，2，8〕，〔8，2，1〕和〔2，1，8〕都是相等的。

如果一个集合的所有成员都是另一个集合的成员，则第一个集合被包含于第二个集合中。例如：

集合〔1，2〕被包含于集合〔1，2，8〕中。

如果两个集合不相等，这种包含称为"严格包含"。集合〔1，2，8〕也包含于集合〔8，2，1〕中，这是不严格包含。

对集合有三种操作，它们类似于整数的加法、乘法和减法运算。

A和B两个集合的和，也称集合的并，写作A＋B，它的成员是A的成员或B的成员。例如：

〔1，2，3〕与〔2，4，5〕的和是〔1，2，3，4，5〕

A和B两个集合的交，写作A＊B，它的成员既是A的成员又是B的成员。例如：

〔1，2，3，4，5〕和〔2，8，5〕的交是〔2，5〕

A与B的差集，也称集合差，写作A－B，它的成员是A的成员，但不是B的成员。例如：

〔3，5，8，9〕减〔3，4，5〕的差是〔8，9〕。

2.集合类型定义

虽然在集合论中对作为集合成员的对象没有限制，但在PASCAL中，只能使用受限制的集合数据类型。一个集合的成员必须都为同一种类型，它就是集合的基类型。基类型必须是简单类型，即除任何实型以外的纯量类型。集合类型可通

过保留字SET OF和一简单类型来定义。例如：

```
TYPE
  FRUITS=(APPLE, BANANA,ORANGE,GRAPE,
                    MANGO, WALNUT, PEANUT,
                    PEAR, PEACH);
  FOOD=SET OF FRUITS;
  CHARACTERS=SET OF CHAR;
  LATTER=SET OF 'A'..'Z';
VAR
  EATABLES1, EATABLES2：FOOD;
        A, CH：CHARACTERS;
```

PASCAL规定一个集合的成员最多512个，基类型的序值必须在0至511。

3.集合的一般形式

在PASCAL中，集合的最一般形式是包含在方括号中用逗号隔开的一系列表达式。例如·

〔'A', 'P', 'P', 'L', 'E'〕

〔1, 2, 3〕

〔X, Y, Z〕

〔1..5〕（等价于集合〔1, 2, 3, 4, 5〕）

〔'A'..'L', 'a'..'z', '0'..'9'〕

〔 〕　　（空集）

4.集合的比较和运算

集合运算符按其优先级分为三类：

（1）*　　集合交

（2）+　　集合并

　　　-　　集合差

（3）=　　集合相等

<>　　集合不等

<=　　集合包含于

>=　　集合包含

IN　　检查集合成员资格。IN左边的运算量是一个表达式，而右边的运算量是一个与它相联系的集合类型表达式。如果左边运算量是右边运算量的一个成员，结果为真，否则为假。

例如：

```
[A,B]=[A,B]              为真
[ICECREAM]<>[ICE,CREAM]  为真
['A'..'Z']<=['A'..'Z','a'..'z',
                 '0'..'9'] 为真
['A'..'Z']>=['A','P','P','L','E'] 为真
APPLE IN [APPLE,PASTRY,SUGER] 为真
APPLE IN [STRAWBERRIES,CREAM] 为假
```

经常使用成员关系检查，对于说明复杂的判别是很有用的。例如：

如果希望将输入的数字字符进行计数，其余的不必计数，可用如下语句：

```
READ(CH);
IF(CH='1')OR(CH='2')OR(CH='3')OR(CH='4')
   OR(CH='5') OR (CH='6') OR (CH='7')
   OR (CH='8') OR (CH='9') OR (CH='10')
   THEN J:=J+1;
```

这样表示语句太长，执行起来慢。也可写成：

```
IF(CH>='0') AND (CH<='9') THEN J:=J+1;
```

但用集合可简化为：

IF CH IN ['0'..'9'] THEN J:=J+1;

5.集合赋值

使用赋值语句，可以将集合表达式的值赋给集合变量。集合表达式由集合常数、集合变量、集合运算符和集合一般形式组成。例如：

```
TYPE ASCII=SET OF 0..127
VAR NOPRINT,PRINT,ALLCHARS:ASCII;
 BEGIN
    ALLCHARS:=[0..127];
    NOPRINT:=[0..31,127];
    PRINT:=ALLCHARS-NOPRINT
 END;
```

我们举 1 个使用集合的例子。

例4　编制一程序，根据用户的需要建立一个集合，然后从这个集合中清除部分元素，最后输出集合中剩余的元素。

```
PROGRAM SETDEMO(INPUT,OUTPUT);
 TYPE M=1..20;
      S=SET OF M;
 VAR X,Y1,Y2:S;
      I:M;
      J:INTEGER;
  BEGIN
   X:=[1..20]; Y1:=[ ]; READ(I);
   WHILE I IN X DO
      BEGIN
        Y1:=Y1+[I];
        READ(I)
      END;
   Y2:=Y1;
    FOR J:=1 TO 15 DO
     IF J IN Y2 THEN Y2:=Y2-[J];
     WRITE('Y1=[');
```

```
FOR I:=1 TO 20 DO
 IF I IN Y1 THEN WRITE(I,' ');
 WRITELN(']');
 WRITE('Y2=[');
FOR I:=1 TO 20 DO
 IF I IN Y2 THEN WRITE(I,' ');
 WRITELN(']')
END.
```

11.5 数组类型

1.数组的定义

一个数组是变量的一个有序集合，其中所有变量具有同样的类型。每个变量可以通过下标来访问。下标是纯量类型的表达式，由方括号之间的下标类型表达式组成。用一个下标类型和一个基类型说明一个数组类型。例如：

```
TYPE DIRECTION=(X,Y,Z);
     VECTOR=ARRAY[DIRECTION] OF REAL;
     NUMBER=ARRAY[1..10] OF INTEGER;
  VAR U,V:VECTOR;
     NUB:NUMBER;
```

VECTOR的下标类型是DIRECTION，基类型是REAL。类型NUMBER的下标类型是子界类型1..10，基类型是整型。

类型VECTOR的每一个变量都有三个分量，和类型DIRECTION的三个值一一对应。变量V的3个分量是V[X],V[Y]，V[Z]。

V的每一个分量是一个实数型变量，它可用在能使用实数型变量的任何地方。例如：

V[X]:=2.1；V[Y]:=3.5；V[Z]:=5.0；

赋值和比较运算允许在任何两个相同类型变量之间进行，因此，可将一数组整个赋值给另一个数组。

2.多维数组

数组的基类型可以是任何数据类型，特别地，它可以是另一个数组，这种结构称作多维数组。例如：

```
TYPE CHESSBOARD=ARRAY[1..8] OF
          ARRAY[1..8] OF CHESSPIECE;
     CUBE=ARRAY[SIZE] OF ARRAY[SISE]
          OF ARRAY[SIZE] OF REAL;
```

一个多维数组可以通过多重下标来方便地定义。上面的定义可重写为：

```
TYPE CHESSBOARD=ARRAY[1..8,1..8] OF
                CHESSPIECE;
     CUBE=ARRAY[SIZE,SIZE,SIZE] OF
                REAL;
VAR  BOARD:CHESSBOARD;
     SOLD:CUBE;
```

因此，下列赋值是合法的。

```
SOLD [1, 1, 1] :=2.1;
BOARD [1, 1] :=BOARD [1, 2];
```

3.字符型数组

字符型数组是带有一个下标，且其元素为标准纯量型字符的数组。在PASCAL里，定长字符串可以用字符的紧缩数组表示。例如：

```
TYPE
  STRING=PACKED  ARRAY [1..20] OF  CHAR;
VAR
  N, CH :STRING;
```

类型STRING的变量可象其它任何数组一样使用。它们

可被赋值并且可进行全部六种关系运算。串变量和串表达式计算出来的值不能赋给字符型数组。例如：

CH：='THIS IS TWENTY CHARS'；

N：='THIS IS FIFTEEN'；

N<CH 为真

N>CH 为假

下面给出两个例子，说明数组的应用。

例5 求两个矩阵M和N的积L。

```
PROGRAM ARRAYDEMO(INPOT,OUTPUT);
 CONST SIZE=10;
 TYPE SUBSCRIPT=1..SIZE;
      MATRIX=ARRAY[SUBSCRIPT,SUBSCRIPT]
             OF REAL;
 VAR M,N,L:MATRIX;A,B:REAL;
     R,S,T:SUBSCRIPT;
 BEGIN
    FOR S:=1 TO SIZE DO
      FOR T:=1 TO SIZE DO
       BEGIN
         READ(A,B);
         (*输入矩阵各元素的数据*)
         M[S,T]:=A;
         N[S,T]:=B;
       END;
    FOR R:=1 TO SIZE DO
       FOR S:=1 TO SIZE DO
         BEGIN
          L[R,S]:=0;
          FOR T:=1 TO SIZE DO
          L[R,S]:=C[R,S]+M[R,T]*N[R,S]
         END;
    FOR R:=1 TO SIZE DO
      FOR S:=1 TO SIZE DO
        WRITE(L[R,S]);

         (*输出矩阵L*)
        WRITELN
  END.
```

例6 将任意几个数按从小到大排列。

```
PROGRAM  SORT(INPUT, OUTPUT);
VAR
  A：ARRAY [1..N] OF REAL;
  N, I, J：INTEGER;
  TEMP：REAL;
BEGIN
  READ(N);
  FOR I：=1  TO N DO
    READ(A [I] );        (输入要排序的数)
  FOR I：=1  TO N-1 DO
    FOR J：=I TO N-1 DO
      IF A [J] >A [J+1] THEN
        BEGIN
          TEMP：=A [J] ;
          A [J] ：=A [J+1] ;
          A [J+1] ：=TEMP
        END;
  FOR I：=1 TO N DO
    WRITE(A [I] , '    ')        (输出结果)
END.
```

11.6 串类型

11.6.1 串变量及其赋值

串是字符组成的序列。由单引号括起来的字符序列称为

串常量。在PASCAL语言中串作为一种数据类型，可以用一维紧缩字符数组来定义：

TYPE STRING=PACKED ARRAY〔1..N〕OF CHAR;

其中N为大于1的整型常数，它定义了一个包含字符个数不超过N的串类型。可以如同使用数组变量一样对串类型的变量赋值及给以下标。

1.串变量说明

在PASCAL中，串是一种预定义类型，由STRING标识。串类型实质上是字符型紧缩数组，它的字符个数可在0到规定的上界之间动态地变化。但串变量和字符型紧缩数组之间不能相互赋值。

串变量定义由保留字STRING和方括号中的最大长度组成。长度规定为0到255之间的整型常量。串变量的默认长度为80个字符。例如：

```
VAR TITLE：STRINC;        （默认长度为80个字符）
    NAME：STRINC〔30〕；   （指定长度为30个字符）
```

检索串变量是从第一个字符至最后一个字符，不检索空串。因此TITLE〔1〕表示串TITLE的第一个字符，TITLE〔LENGTH（TITLE）〕是串TITLE的最后一个字符。

2.串表达式

串表达式由串常量、串变量、函数命名符和运算符构成。

“+”可用来联接动态串。CONCAT函数的功能和它一样。例如：

'APPLE'+'PASCAL'='APPLEPASCAL'

'A'+'B'+'C'='ABC'

'12'+'.'+'34'='12.34'

关系运算符=、<、>、<>、<=、>=的优先级低于联接运算符。当比较两个串时，是从左至右按照它们的ASCII值逐个比较。如果两个串前面部分相同，则长的串大于短的串；如果两个串从第i个字符开始不同，则ASCII值小的第i个字符所在的串小于另一个串；仅当两个串的长度和内容对应相同时它们才相等。例如：

'A'<'B'	真
'123'<'12345'	真
'123'<'42'	真
'APPLE'<'APPLE'	假
'APPLE'='APPLE'	真

3.串赋值

赋值运算符可用来将串变量、串常量、串标准过程和函数赋值给串变量。例如：

```
AGE:='18';
NAME:='PETER';
NAME1:=COPY(TITLE, 1, 30);
```

串亦能被说明为常量。例如：

```
CONST  SAMMY='HI THERE';
```

注意：不能对长度为0的串进行检索，否则产生错误。为避免此种错误，可先用LENGTH函数检测其长度。

仅含一个字符的串值和CHAR类型的变量值不是同一回事，它们属于不同的数据类型。但仅包含一个字符的串常量和CHAR类型常量，正好具有相同的形式，这种常量既可作CHAR类型的变量值，又可作串值来使用。

11.6.2 串标准函数和过程

1.长度函数

格式：LENGTH (STRG)

这个函数返回串表达式STRG的长度．即串中字符的个数，其结果类型为整数。如：

若GEESTRING:='1234567'

则LENGTH (GEESTRING) 返回值为7。

2.求子串位置函数

格式：POS (SUBSTRG, STRG)

SUBSTRG和STRG可以是串表达式或串变量。POS函数寻找STRG串变量中第一次出现的SUBSTRG串变量的值的开始位置，并返回该位置值。串中第一个字符位置是1。如果没有找到，则返回值为0。例如：

```
STUFF:='TAKE THE BOTTLE WITH A  METAL
            CAP'
PATTERN:='TAL';
POS(PATTERN, STUFF)          (返回值为26)
POS('X', STUFF)              (返回值为0)
```

3.字符串连接函数

格式：CONCAT (STR1, STR2, …, STRN)

将串表达式STR1, STR2, …STRN连接成一个新的字符串。例如：

```
ST1:='THIS IS A STRING';
ST2:='THIS IS A LONG STRING';
```

CONCAT (ST1, '—', ST2) 返回值为'THIS IS A STRING—THIS IS A LONG STRING'。

注意：STR1，STR2，…，STRN可以是串常量，也可以是有值的串变量。

4.取子串函数

格式：COPY（STRG，INDEX，COUNT）

STRG为串表达式，INDEX和COUNT为整型表达式。该函数返回STRG中从INDEX开始的COUNT个字符所组成的一个子串。如果INDEX超出了串的长度，则返回一个空串。例如：

```
TL:='KEEP SOMETHING HERE'
COPY(TL, 6, 9)        (返回'SOMETHING')
COPY(TL, 16, 4)       (返回'HERE')
COPY(TL, 21, 2)       (返回'    ')
```

5.删除子串过程

格式：DELETE（STRG，INDEX，COUNT）；

其中STRG为串变量，INDEX和COUNT是两个整型表达式。DELETE删除STRG中从INDEX开始的COUNT个字符的子串。删改后的串仍放在串变量STRG中，经它回送给调用程序段。例如：

```
ST1:='ABCDEFGH';
DELETE(ST1,2,4); (*结果ST1:='AFGH'*)
OVERSTUFFED:='THIS STRING HAS FAR TOO
          MANY CHARACTERS IN IT.';
DELETE(OVERSTUFFED,POS('HAS',
        OVERSTUFFED)+3,8);
WRITELN(OVERSTUFFED)
```

将输出：

THIS STRING HAS MANY CHARACTERS IN IT.

6.插入子串过程

格式：INSERT（SUBSTRG，STRG，INDEX）；

SUBSTRG为串表达式，STRG为串变量，INDEX为整型表达式。INSERT把SUBSTRG的值插入到串STRG的INDEX位置上。例如：

ST1：='ABCDEFG'，则INSERT（'ZZ'，ST1,3）；的结果为'ABZZCDEFG'。

7.将整型转换成串类型

格式：STR（LONG，STRG）；

LONG为整型值，可为长整数型。STRG为串变量。STR过程将LONG的值转换成一个串后存放于STRG中。例如：

```
I:=1234;  STR(I,STR1)给STR1的值为'1234'
INTLONG:=123456789;
STR(INTLONG,INTSTRING);
 给INTSTRING的值为'123456789'
INSERT('.',INTSTRING,
          LENGTH(INTSTRING)-1);
WRITELN('$',INTSTRING)
```

将输出：$1234567.89

最后，作为串标准过程和函数应用的一个例子，给出下面字处理程序。

例8　找出句子中的单词，以空格来区分单词。

本程序对用户键入的句子（80个字符长），逐个输出句子中以空格分隔的单词。

```
PROGRAM SCANNER:
CONST SPACE=' ';
VAR PHRASE:=STRING:
    FIRST:INTEGER;
BEGIN
WRITELN('TYPE IN A SENTENCE AND PRESS
                RETURN');
READLN(PHRASE);
WRITELN;
```

```
PHRASE:=CONCAT(PHRASE,SPACE);
WHILE LENGTH(PHRASE)>1 DO
   BEGIN
    FIRST*=POS(SPACE,PHRASE);
    WRITELN(CQPY(PHRASE,1,FIRST-1));
    DELETE(PHRASE,1,FIRST)
   END
END.
```

习题十一

1.为什么要使用纯量类型数据？纯量类型数据有什么特点？

2.假设定义 TYPE COIN=(PENNY, NICKEL, DIME, QUARTER, HALFDOLLAR)；求下列各表达式的值。

```
PRED(PENNY) PRED(DIME) PRED(HALFDOLLAR)
SUCC(PENNY) SUCC(QUARTER)
SUCC(HALFDOLLAR)
ORD(PENNY)  ORD(NICKEL)  ORD(QUARTER)
PENNY<DIME  NICKEL=DIME
QUARTER>=HALFDOLLAR
```

3.使用子界类型数据有什么好处？有哪些地方经常使用这种数据类型？

4.定义TYPE COUNTER=1..20;

VAR A, B, C：COUNTER;

下列语句哪些是正确的，哪些是错误的？

A：=200+100; B：=100+20; C：=B−30;

B：=PENNY; A：=31.5; C：='A';

5.定义VAR X, Y, Z：INTEGER [25];

A, B：INTEGER;

D, E：REAL;

下列语句哪些是正确的，哪些是错误的？

A：=X+Y; X：=−Y; A：=TRUNC(Z);

A：=D+Y; E：=Z; X：=D+E;

Y：=B+D；D：=B−E；Y：=Z−E；

6.求下列表达式的值。

```
'12AB'<'AB12'   '12'<'1211'
'APPLE PASCAL'='APPLEPASCAL'
'TIME10'<='TIME10'
```

7.定义VAR TITLE：STRING〔30〕；

A：CHAR；

下列语句中哪些是正确的，哪些是错误的？

```
TITLE:='18-3-89';   TITLE[5]:='PETER';
TITLE:=' ';         TITLE:=21+'A':
TITLE[1]:='A';      A:=TITLE[1];
A:='B';  A:='BC';   TITLE:=A;
```

8.定义VAR S，T：STRING；求下列表达式的值。

```
'LENGTH('APPLE PASCAL')
 LENGTH('THIS IS A PEN')
 POS('TH','THIS IS A PEN')
 POS('A' 'THIS IS A PEN')
 CONCAT('TURBO','PASCAL',LANGUAGE')
 CONCAT('THE NUMBER OF','ZERO','IS',
       'COUNT')
 COPY('TOTAL OF POSITIVE IS
                COUNT',10,8)
 DELETE('TOTAL OF NEGATIVE IS
                 COUNT',6,12)
```

若

```
S:='DOES SHE ENJOY RIDING?';
  T:='DOES ENJOY RIDING?'
  INSERT('NO',S,23)
  INSERT('SHE',T,6)
  STR('8192',S)
  STR('100012',S)
```

9.给出TYPE NUMBERSET=SET OF 1..15；编一程序，打印出NUMBERSET的变量的值。例如：

其成员为3、8、9和11的集合，应打印出〔3，8，9，11〕。

10.输入一系列字符，将数字字符，英文字母字符和其它字符分别计算，并求出各占的百分比，输出格式要求如下：

NAME	NUMBER	PERCENT
LETTER	?	?%
DIGIT	?	?%
OTHER	?	?%

遇到"！"停止计数，用集合类型。

11.建立两个集合，然后对它们进行并、交和差运算，最后输出运算的结果。

12.列出下列集合的全体元素。

(1) SET OF 〔RED, GREEN, BLUE, WHITE〕；

(2) SET OF 1..8；

(3) SET OF 'A'..'P'；

13.求下列集合表达式的值。

(1) 〔1..5〕*〔3, 5〕

(2) 〔1..4〕+〔3, 4〕

(3) 〔2..8〕-〔3..8〕

14.下列布尔表达式中哪些得出真值？

(1) 〔1, 2, 3〕=〔1..3〕

(2) 〔1, 2, 3〕<=〔1..3〕

(3) 3 IN 〔4, 5, 6, 7〕

(4) 'A' IN 〔'P', 'X', 'Y', 'A'〕

15.某本书可由下述说明定义：

```
CONST LINELEN=70;
      PAGESIZE=55;
      THICKNESS=330;
TYPE  LINE=ARRAY [1..LINELEN] OF CHAR;
```

```
      PAGE=ARRAY [1..PAGESIZE] OF LINE;
      VOLUME=ARRAY [1..THICKNESS] OF
                   PAGE;
VAR  BOOK: VOLUME;
```

指出下述每一表达式的类型和值：

BOOK [25], BOOK [187] [50], BOOK [46][7] [10]

给一个合适的格式，并编写打印BOOK内容的程序。

16. 定义两个数组类型，其下标分别为子界和纯量类型，基类型分别为整数和实数类型。建立一个包含十项由整数元素构成的数组，然后将这个数组中间的六项内容依次传送给另一个包含六项由实数元素构成的数组，输出后一个实数数组。

第十二章　记录、文件和指针类型

12.1　记录类型

一个记录是有若干分量的一种结构变量。一个记录的分量可以有不同的类型，而且它们是通过名字访问的。

1.记录类型定义

记录类型定义由保留字RECORD、域（又称字段）表以及终止符保留字END组成。域表由一系列用分号隔开的记录组成。每一个记录都由一个或多个用逗号分隔的标识符、冒号和类型组成。因此，每个记录规定了一个或多个域的类型和标识符。例如：

```
TYPE  FULLNAME=RECORD
            SURNAME:STRING[16];
            NOOFFNAMES:1..3;
            FORENAMES:STRING[16]
            END;
         DATE=RECORD
            DAY:1..31;
            MONTH:(JAN,FEB,MAR,APR,MAY,JUN,
               JULY,AUG,SEP,OCT,NOV,DEC);
            YEAR:1901..2000;
            END;
VAR   BIRTH:DATE;
      BOY:FULLNAME;
```

其中FULLNAME类型由SURNAME和FORENAME

两个元素组成。

这两个元素都属于串类型。

DATE类型由三个元素组成：DAY，MONTH和YE-AR。

这三个元素属于三种不同的数据类型：

DAY属于子界类型；

MONTH属于纯量类型；

YEAR属于整数类型

假设生日BIRTH表示1949年10月1日，男孩BOY表示TOM ADDYMAN。可以通过如下赋值语句来实现：

```
BIRTH.DAY:=1;        SURNAME:='ADDYMAN';
BIRTH.MONTH:=OCT; FORENAME:='TOM';
BIRTH.YEAR:=1949
```

从这些赋值语句可以看出，记录元素是通过变量名与域名来表示的。应把变量名与域名用一个圆点连起来。

APPLE PASCAL支持标准PASCAL的紧缩记录类型。例如：下面是一个具有4个域的记录说明，由于将它们说明为紧缩记录，因此整个记录只占用一个16位的存储字。

```
TYPE  R=PACKED  RECORD
            I, J, K:0..31;
                B:BOOLEAN
            END;
```

另外，APPLE PASCAL对记录类型说明有两点限制：

（1）空域是非法的（即RECORD END；），将出现语法错。

（2）变体括号END前的分号也要导致错误。

2.WITH语句

利用WITH语句，可以象简单变量那样来访问记录。WITH语句的格式如下：

WITH〈变量表〉DO〈语句〉

WITH语句就象“打开一个记录”，使得域名就象变量名一样使用。例如：

让BIRTH记录变量表示1949年10月1日，可以用下述语句：

```
WITH BIRTH DO
BEGIN
  DAY:=1;
  MONTH:=OCT;
  YEAR:=1949
END;
```

这表示，打开BIRTH的“疆域”，YEAR，MONTH和DAY都获得了“解放”，书写和阅读都方便多了。

3.记录变体

记录类型的语法提供了一种变化的记录结构，它允许一个记录可以包含一个固定部分，一个变体部分或者包含两者。但如果包含两者，固定部分必须在前面。变体部分包含域表，因此变体记录可以嵌套。

例如考虑三种几何形状：矩形、三角形和圆。矩形可用高和宽来描述：

```
TYPE  RECTANGLE=RECORD
      HEIGHT:REAL;
      WIDTH:REAL;
      END;
```

然而三角形不能用这个方法来描述。对于三角形，合适的构造类型是：

```
TYPE  TRIANGLE=RECORD
                SIDE1,SIDE2:REAL;
                ANGLE:REAL
                END;
```

只要给出圆半径就够了。类型GEOMETRICFIGURE（几何图形）就可以变体记录来定义：

```
TYPE  FIGURE=(RECT,TRI,CIRC);
  GEOMETRICFIGURE=RECORD
      KIND:FIGURE;(*固定部分*)
      CASE FIGURE OF (*变体部分*)
      RECT:(WIDTH,HEIGHT:REAL);
      CIRC:(RADIUS:REAL);
      TRI:(SIDE1,SIDE2,ANGLE:REAL)
      END
```

CASE在条件语句中的选择因子是一个变量，而在记录说明中的情况选择因子则是一个类型。在这个例子中，CASE选择因子是类型FIGURE，该类型只能为纯量类型。

下面给出一个函数，该函数的结果为几何图形的面积。对每一种图形，所涉及的实际计算是完全不同的。CASE语句将用来为每一个变体选择恰当的计算。

```
FUNCTION AREA(FIG:GEOMETRICFIGURE):REAL;
 CONST  PI=3.1415926535;
 BEGIN
  WITH FIG DO
   CASE KIND OF
   RECT:AREA:=HEIGHT*WIDTH;
   CIRC:AREA:=PI*SQR(RADIUS);
   TRI:AREA:=0.5*SIDE1*SIDE2*SIN(ANGLE)
   END
 END;
```

例1 设一个矩形的长为4，宽为5，调用上面的函数，其程序如下：

```
PROGRAM RECORDDEMO;
TYPE FIGURE=(RECT,TRI,CIRC);
  GEOMETRICFIGURE=RECORD
                 KIND:FIGURE;
                 CASE FIGURE OF
                 RECT:(WIDTH,HEIGHT:REAL);
                 CIRC:(RADIUS:REAL);
                 TRI:(SIDE1,SIDE2,ANGLE:REAL)
                 END;
VAR RECTFIG:GEOMETRICFIGURE;A:REAL;
 FUNCTION AREA(FIG:GEOMETRICFIGURE):REAL;
       .
       .  (*上面函数的内容*)
       .
  END;
  BEGIN
   WITH RECTFIG DO
    BEGIN
     KIND:=RECT;
     HEIGHT:=4;
     WIDTH:=5;
     A:=AREA(RECTFIG);
    END;
   WRITE('RECTANGLE AREA IS',A);
  END.
```

12.2 文件类型

文件是由相同类型的元素序列组成的。任何时候只有文件的当前成份需要存于主存储器中，文件本身通常保存在计算机的外存储器上（如磁盘、磁带等）。

对于每一个在PASCAL程序中说明的，取名为F的文件，系统自动引入一个具有文件成份类型的缓冲区变量，记为F∧，它代表当前涉及的文件的一个记录。用文件指针表示

指向当前的文件记录，任何时候只有文件指针所指示的文件记录才是可以访问的。

对于文件类型或任何其它具有文件成份的类型，赋值运算和相等关系是无效的。

12.2.1 文件类型定义

文件类型的定义如下：

FILE OF〈类型〉

例如：

```
TYPE  INTFILE=FILE OF INTEGER;
 PRODUCTNAME=STRING[80];
 PRODUCT=RECORD
         NAME:PRODUCTNAME;
         ITEMNUMBER:REAL;
         INSTOCK:REAL;
         MINSTOCK:REAL;
         SUPPLIER:INTEGER;
         END;
 VAR PRODUCTFILE:FILE OF PRODUCT;
```

文件元素的类型可以为除文件类型之外的其它类型，但最常用的是标识符。

12.2.2 文件操作

下面叙述文件的打开和关闭过程以及指示是否已达到指定文件末尾的函数。标识符FILEID在下面都表示已说明的一个文件变量标识符。

1.REWRITE过程

格式：REWRITE(FILEID, TITLE)；

TITLE是产生任何合法文件名的字符串。文件变量

FILEID被赋以文件名，以后所有关于FILEID的操作就是对磁盘文件TITLE的操作。本过程常用来建立一个新文件，并打开这个文件准备对它输入输出。它也可以用来打开已经存在的文件，但这样要破坏该文件而建立起一个新的文件目录。

2.RESET过程

格式：RESET（FILEID，TITLE）；

或RESET（FILEID）；

TITLE是文件名。该过程为读和写打开一个现存的文件。

对于第一种调用，如果指定的文件已经打开，则发生I/O错，文件状态保持不变；如果文件不存在，也发生I/O错。

第二种调用仅用于一个已打开的文件，它将文件指针指到文件刚被打开时的位置。

如果不是交互文件，则RESET自动完成一个GET操作，即将文件的第一个记录装入文件缓冲区变量，并将文件指针指向第二个记录。如果是交互文件，则不执行GET操作，此时文件缓冲区变量的值没有定义，文件指针指向第一记录。

3.CLOSE过程

格式：CLOSE（FILEID [，OPTION]）；

OPTION为下面四种选择之一：

（1）NORMAL：执行一个正常的关闭，即简单地置文件为关闭状态。如果文件是用REWRITE打开又是一个磁盘文件的话，则它从磁盘文件目录中删除。

（2）LOCK：如果磁盘文件是由REWRITE打开的，

则文件被永久地加入磁盘目录中，并在最后一个文件分量后置文件结束标志。否则执行一个NORMAL关闭。如果与文件名相配的是一个现存的磁盘文件，则此文件原先的内容被丢失。

（3）PURGE：如果文件是磁盘文件，则它从磁盘目录中删除。如果磁盘文件已经存在，并且是用REWRITE打开的，那么原先的文件保留在磁盘目录中不变。如果不是磁盘文件，则相连的设备脱机。

（4）CRUNCH：和LOCK相似，只是将文件结束标志锁在最后一次存取的位置上，即最后一次被访问的文件记录后的任何内容均被抛弃。如果与文件名相配的是一个现存文件，则此文件原先的内容也被丢失。

若省略OPTION，则执行NORMAL关闭。

无论哪种选择，都将关闭标志文件，使得文件缓冲区变量FILEID∧没有定义。

程序结束前总是要关闭打开了的文件，否则没有关闭的文件变成不确定状态。

4.EOF函数

格式：EOF［（FILEID）］

如果省去（FILEID），则默认为INPUT文件。

如果文件指针指在磁盘文件尾部，即最后一个文件元素以外，则EOF为真，否则为假。

文件被打开后，EOF为假，文件关闭后EOF为真。无论何时只要EOF(FILEID)为真，则FILEID∧没有定义。

GET操作后，若GET试图访问一个文件末尾后的记录，则EOF为真。键盘读时，仅当键入文件结束符CTRL—C（ASCI I3）时EOF才为真。

12.2.3 类型文件

类型文件是由变量说明指定了类型的文件。PASCAL为类型文件提供以下三个特殊过程。

1.GET过程和PUT过程

FILEID是一个已说明为类型文件的标识符。

这两个过程分别用于从一个类型文件中读入逻辑记录和将一个逻辑记录写到类型文件上。

GET（FILEID）把文件当前所指的记录内容赋给缓冲区变量FILEID∧，并将文件指针推到下一个文件记录位置。执行GET操作时，若下一个文件记录不存在，则EOF为真，FILEID∧中的结果值无定义，故GET的作用仅在执行前EOF为假时才有定义。

PUT（FILEID）把FILEID∧的内容写到文件当前所指的位置，并将记录指针推到下一个记录位置。下一个使用相同FILEID的GET或PUT操作，将顺序地访问文件的下一个记录，直到下一次物理上相联系的盘块不再作为当前工作块时，才可对磁盘进行实际存取。强迫块被写的操作类是：对文件任何记录的SEEK（查找）。RESET和CLOSE当越过块边界时，对文件的相继GET或PUT操作，将引起I/O错。

下面用几个简单例子来说明REWRITE、RESET、CLOSE、EOF、GET和PUT的使用方法。

例2.建立一个实数文件。

```
PROGRAM MAKEFILE;
VAR F:FILE OF REAL;
    I:INTEGER;
 BEGIN
```

```
 REWRITE(F,'APPLE:REALS,DATA');
    (*打开新文件，准备写*)
 FOR I:=1 TO 10 DO
    (*读入10个实数，存入文件*)
   BEGIN
     WRITE('-->'); (*置一提示符于屏幕*)
     READ(F^); (*键盘读入实数到F^中*)
     PUT(F) (*将该数存入文件*)
   END;
 CLOSE(F,LOCK)
END.
```

该程序运行后，在APPLE 1:盘上建立一个新的文件REALS.DATA，它有10个实数分量。

例3 读文件数据到屏幕。

```
PROGRAM READFILE;
VAR F:FILE OF REAL;
BEGIN
 RESET(F,'APPLE1:REALS.DATA'); (*打开文件*)
 WHILE NOT EOF(F) DO  (*读取数据*)
    BEGIN
     WRITE(F^);  (*显示当前数据*)
      GET(F)  (*前进到下一个数据*)
    END;
  CLOSE(F)  (*关闭文件*)
END.
```

本程序是将例2建立的文件数据读到屏幕上显示出来。

例4 文件复制。

```
PROGRAM COPYFILE;
TYPE IMAGE=PACKED ARRAY[1..16,1..14] OF
           BOOLEAN;
     IMAGEFILE=FILE OF IMAGE;
VAR F,G:IMAGEFILE;
BEGIN
  RESET(F,'APPLE1:FF.DATA');
  REWRITE(G,'WORKDISK:GG.DATA');
WHILE NOT EOF(F) DO
```

```
BEGIN
    G^:=F^;
    PUT(G); GET(F)
END; (*WHILE*)
CLOSE(F);  CLOSE(G,LOCK)
END.
```

该程序将APPLE1：盘上的FF.DATA数据文件复制到WORKDISK：盘上，并以名GG.DATA存盘。

2.SEEK过程

格式：SEEK（FILEID，RECNUM）；

FILEID是一个已打开的类型文件标识符，RECNUM是一个解释为文件记录号的整数值。

本过程改变文件指针，使得对文件的下一个GET或PUT用到由RECNUM所指明的FILEID的记录。文件中的记录从0开始编号。两个SEEK调用之间须执行一个GET或PUT，否则会造成意想不到的后果。SEEK执行后EOF返回假，后跟GET或PUT将使EOF回到适当值。

例5 用SEEK来随机存取和修改文件记录。

```
PROGRAM RANDOMACCESS(INPUT,OUTPUT);
VAR RECNUMBER:INTEGER;
    FNAME:STRING;
    VITALS:FILE OF RECORD
         NAME:STRING[20];
         DAY,MONTH,YEAR:INTEGER;
         ADDRESS:STRING[50];
         ALIVE:BOOLEAN
       END;
BEGIN
WRITE('ENTER FILENAME');  (*获得文件名*)
READLN(FNAME);
(*$I-*) (*暂停I/O检查，使FNAME不存在时，
       不致出错*)
RESET(VITALS,FNAME);
```

```
    (*用RESET保护文件现有内容*)
IF IORESULT<>0 THEN REWRITE(VITALS,FNAME);
 (*IORESULT不等0,文件不存在,
                    用REWRITE打开*)
(*$I+*)  (*恢复I/O检查*)
WHILE TRUE DO (*TRUE为真，重复执行,
                只有EXIT执行才退出程序*)
  BEGIN
    WRITE('ENTER RECORD NUMBER');
              (*请键入记录号*)
    READLN(RECNUMBER);  (*得到记录号,
                  键入CTRL-C，则EOF为真*)
    IF EOF THEN
       BEGIN
         CLOSE(VITALS,LOCK);
         EXIT(PROGRAM)  (*退出程序*)
       END;
    SEEK(VITALS,RECNUMBER);
           (*得到指定的记录*)
    GET(VITALS);
    WITH  VITALS^ DO (*修改记录*)
      BEGIN
        WRITELN(NAME),
        WRITE('ENTER CORRCT NAME');
        READLN(NAME);
        WRITELN(DAY);
        WRITE('ENTER CORRCT DAY');
        READLN(DAY);
        (*修改记录的其它域....*)
      END;
```

（下面SEEK同一记录。因为当前记录进入VITALS入以后，GET操作已将文件指针推进到下一记录处）

```
SEEK(VITALS,RECNUMBER);
PUT(VITALS);  (*修改的记录写入文件,
               若EOF真，退出程序*)
IF EOF(VITALS) THEN
 BEGIN
```

```
      WRITELN('NOT ENOUGH FILE SPACE');
      EXIT(PROGRAM)
     END
    END
   END.
```

12.2.4 文本文件和交互文件

1.文本文件

文本文件不是某一类型值的序列，这是与其它文件类型的不同之处。一个文本文件的基本元素是字符，它们构成行，各行由行结束符↩来标志（ASCII 13），文件结束标志为CTRL-C（ASCII 3）。行的长度是变化的，因而某一字符的位置是无法计算的。所以，文本文件只能顺序处理，对文本文体不能同时执行输入输出操作。

使用保留字TEXT来说明一个文本文件变量。为了读而打开一个磁盘文本文件，先调用RESET过程；为了写而打开一个磁盘文本文件，先调用REWRITE过程。

2.文本文件操作

PASCAL为文本文件和交互文件提供了以下五个专用过程和一个函数。

（1）READ过程

格式：READ（［FILEID，］VBLS）；

FILEID为已打开的文本文件或交互文件标识符。若FILEID省略，就默认为INPUT文件。VBLS是由逗号分开的一个或多个变量。变量可为字符、串、整数、长整数和实数类型。对串变量应使用READLN。

本过程从文件中读值并将它们依次赋给变量。

对于字符变量，READ从文件中读取一个字符并把它赋

给该变量。当读到行结束符↵时，则将空格符␣(ASCII 32)赋给该变量。当EOF为真时，赋给字符变量的值没有定义。一次READ后，下一次READ或READLN时，总是从被读取的字符的下一个字符开始。对于文本文件，读到文件的最后一个字符时，EOF为真；当读到一行的最后一个字符或EOF为真时，EOLN为真。对于交互文件，读到文件结束符时EOF为真；当读到一行的结束符或EOF为真时，EOLN才为真。如想读取一行文本，要用EOLN检查是否已读到一行的结尾。但用带有串变量的READLN，就不必检查EOF而进行整行读。

对于数值类型变量，READ期待读取一串能解释为同类型数值的字符。位于该串前的空格、行结束符总是被跳过。该串后面总应有一个空格、行结束符或文件结束符。若跳过若干空格和行结束符后找不到数字串，则发生I/O错。否则该串被转换成数值赋给变量。

每次READ后，再次READ时，则从刚被读取的数字串的最后一个字符的后一个字符开始。若数字串的最后一个字符为一行的最后一个字符，则EOLN为真；若为文件最后一个字符，则EOF和EOLN同时为真。若EOF前除了空格外没别的，读取数字则为0。

不管文本类型还是交互类型，带数值变量的READ动作是相同的。

(2) READLN过程

格式：READLN([FILEID,] VBLS)；

该过程允许按行来读取字符和数值变量。参量意义和READ相同。

READLN的工作和READ相似，不同的只是最后一个

变量被读入值后，行上剩下的内容被跳过。READLN执行后，如有后继行，下一个READ或READLN将从下一行第一个字符开始，如没后继行，EOF为真。

对于串变量，READLN将读入行结束符前的所有字符，但不包括行结束符。重复执行带串变量的READLN，将起到把文件的相继行作为串来读入的作用。

（3）WRITE和WRITELN过程

这两个过程将字符、串和数值变量的值写进文件，WRITELN还进一步允许按行输出。

格式：WRITE（[FILEID，][ITEMS]）；

WRITELN（[FILEID，][ITEMS]）；

FILEID为已打开的文本文件或交互文件标识符，若省略它，则默认为OUTPUT。ITEMS是用逗号分隔开的一个或多个项目。每个项目可取下列形式之一：

EXPR

EXPR：FIELDWIDTH

EXPR：FIELDWIDTH：FRACTIONLENGTH

其中EXPR是被写到文件的表达式。FIELDWIDTH是一个整型表达式，指出被写字符的最少个数。FRACTIONLENGTH是整型表达式，当EXPR为实型时，指出小数点后的数字个数，它的默认值为5，FIELDWIDTH的默认值为1。对于正实数，第一个数字前总写上一个空格；负实数，负号占据该位置。

WRITE对表达式求值，并把它们的值顺序地写到文件中。若EXPR具有数值类型，WRITE按PASCAL的数格式将EXPR的值转换成字符串写到文件并将指针指向下一个元素。若EXPR具有字符、串或紧缩数组类型，WRITE将字

符写到文件并将指针指向下一个元素。

例6　逐行读入文件，移去每行中的前导句号，并将它们写到输出文件中。

```
PROGRAM FLUSHSPERIOD;
CONST  PERIOD='.';
VAR INFILE,OUTFILE:TEXT;
  INNAME,OUTNAME,LINEBUF:STRING;
BEGIN
 WRITE('NAME OF INPUT FILE:');
 READLN(INNAME);
 IF POS('.',INNAME)=0 THEN
          (*若未键入扩展名，加上.TEXT*)
  INNAME:=CONCAT(INNAME,'.TEXT');
 (*$I-*)  (*关闭I/O检测*)
  RESET(INFILE,INNAME);
          (*打开已存在的文件*)
  IF IORESULT<>0 THEN
          (*IORESET<>0,文件不存在*)
    BEGIN
      WRITELN('FILE NOT FOUNT');
      EXIT(PROGRAM)
    END;
 (*$I+*)  (*恢复I/O检测*)
 WRITE('NAME OF OUTPUT FILE:');
 READLN(OUTNAME);
 IF POS('.',OUTNAME)=0 THEN
 OUTNAME:=CONCAT(OUTNAME,'.TEXT');
 REWRITE(OUTFILE,OUTNAME);
 (*移去每行前导空白后写到OUTFILE*)
 WHILE(NOT EOF(INFILE))AND(NOT EOF(OUTFILE))
        DO
   BEGIN
     READLN(INFILE,LINEBUF);
     IF LENGTH(LINEBUF)>0 THEN
     IF POS(PERIOD,LINEBUF)=1 THEN
                       DELETE(LINEBUF,1,1);
     WRITELN(OUTFILE,LINEBUF)
   END;
```

```
IF EOF(OUTFILE) THEN
      (*输出文件不完全将它丢弃*)
  BEGIN
    WRITE('NOT ENOUGH ROOM IN
                   OUTPUT FILE !');
    CLOSE(OUTFILE,PURGE)
  END
 ELSE CLOSE(OUTFILE,LOCK)
CLOSE(INFILE)
END.
```

（4）PAGE过程

格式：PAGE（FILEID）；

FILEID为一打开的文本文件或交互式文件。该过程将一个页顶字符（ASCII 12）送给文件。

（5）EOLN函数

格式：EOLN（FILEID）

本函数回送一个布尔值，指示文件指针是否在行的末尾。如果省略FILEID，则默认为INPUT文件。。文件打开后，EOLN立即为假。文件关闭后EOLN为真。当GET找到一个行结束符⌉时，EOLN为真，此时把一空格符（ASCII 32）而不是行结束符装入文件缓冲区变量。

3.交互文件

和文本文件一样，交互文件也是一种字符文件，区别在于交互文件处理RESET、READ和READLN的方式和文本文件不同。使用保留字INTERACTIVE可以说明一个交互文件变量。

当一个PASCAL程序从一个文本文件读字符时，首先必须用RESET打开此文件。RESET自动地执行一个GET

操作，即把文件的第一个字符送入文件缓冲区变量，然后文件指针指向下一个字符。对于随后带来字符类型变量的READ或READLN语句，其工作过程是先把存放在文件缓冲区变量中的字符赋给该字符型变量，然后执行一个GET操作。

如果是交互文件，则用RESET打开文件时不执行GET操作，缓冲区变量没有定义，文件指针仍指向文件的第一个字符。因此，要使READ或READLN能读到字符，须先安排一个GET操作，然后取出GET放入缓冲区变量中的字符。这与文本文件的READ执行顺序正好相反。

换句话说，若CH为字符型变量，INFILE为TEXT型文件，则READ（INFILE，CH）等价于

CH:=INFILE∧;

GET（INFILE）;

若INFILE为INTERACTIVE型文件则READ（INFILE，CH）等价于

GET（INFILE）;

CH:=INFILE∧;

使用交互类型文件的主要原因是，象键盘这样的设备，字符没有键入之前是不可能对它进行GET的。因此当程序从键盘上读取字符时，TEXT(文本)类型文件会遇到麻烦。因为启动程序后，系统用RESET自动打开INPUT文件，假如INPUT具有TEXT（文本）类型，那么一个GET将等待从键盘上接收字符。当键入某字符如A时，字符A送文件缓冲区变量，进入READ语句，程序又等待输入。直到键入第二个字符时，第一个字符A才显示出来。定义INPUT为INTERACTIVE类型文件就避免了这种麻烦。因为对于

INTERACTIVE类型，直到程序执行READ或READLN时才执行GET操作。

例7　该程序用来说明文本文件和交互文件的细微区别。

```
PROGRAM SKETCH;
VAR X:CHAR; F:TEXT;
BEGIN
 REWRITE(F,'WORKDISK:H。DATA')
 F^:='A';
 PUT(F);
 CLOSE(F,LOCK);
 RESET(F,'WORKDISK:H.DATA');
 WHILE NOT EOF(F) DO
  BEGIN
    READ(F,X);
    WRITELN(X)
  END;
 CLOSE(F)
END.
```

程序中F定义为文本文件，第5～8行完成将字母A送到文件F，作为它的唯一文件记录，并关闭F。第9行重新打开与F相联系的文件。第10～15行完成F的记录读入到屏幕并关闭它。结果在屏幕上显示字母A。

将上面程序中的F改为交互类型文件，其它均不作改变（即将第3行的F：TEXT；改为F：INTERACTIVE；）。重新运行该程序，结果在屏幕上显示两行单个字母A。

这是什么原因？当F为交互类型文件时，第9行的RESET语句并不自动执行一个GET操作，文件指针指向F的第一个记录，EOF（F）为假。当执行到READ语句时，对于交互文件是先执行GET操作，将当前指向的记录即字母A送缓冲区变量F∧，文件指针指向下一个元素。然后执行X：＝F∧，这样第13行的写语句就将第一个字母A输出到屏幕。此

时，EOF（F）继续为假，第二次执行READ语句，这次GET造成EOF（F）为真，留下F∧处于没有定义状态。APPLE PASCAL处理这种没有定义的状态，是保留F∧等于它的前一个值，就是字母A，因此不正确地印出了第二个字母A。

12.2.5 无类型文件

无类型文件只要用FILE来说明，无须指明文件的成份类型。例如：

VAR F:FILE;

定义F为一个无类型文件变量。

无类型文件只能用于为实现高速数据传送的内部函数BLOCKREAD和BLOCKWRITE。

一个无类型文件可以看作是一个没有文件缓冲区变量F∧的文件。这种文件的所有输入/输出必须由函数BLOCKREAD和BLOCKWRITE来完成。

这两个函数用于对无类型文件传送数据。它们回送一个表示被传送的数据块数的整数值。

格式：BLOCKREAD（FILEID，ARRAYNAME，BLOCKS[，RELBLOCK]）

BLOCKWRITE（FILEID，ARRAYNAME，BLOCKS[，RELBLOCK]）

其中FILEID是已说明为无类型文件的标识符。ARRAYNAME为说明了的数组标识符，数组长度是512的整数倍，ARRAYNAME可带下标以指出数组中的起始位置。BLOCKS是被传送的块数。RELBLOCK是相对于文件开头的块号。0块为文件的第一块。如果没有出现RELBLO-

CK，则进行顺序地读/写。

使用这两个函数应特别小心，因为没有注意数组的界。当文件的最后一块被读时，EOF为真。

例8　应用BLOCKREAD和BLOCKWRITE的一个例子。

```
PROGRAM FILEDEMO;
VAR G,F:FILE;
  BADIO:BOOLEAN;
  BLOCKNUMBER,BLOCKSTRANSFERRED:INTEGER;
  BUFFER:PACKED ARRAY[0..511] OF CHAR;
BEGIN
  BADIO:=FALSE;
  RESET(G,'SOURCE.DATA');
  REWRITE(F,'DESTINATION');
  BLOCKNUMBER:=0;
  BLOCKSTRANSFERRED:=BLOCKREAD(G,BUFFER,1,
                                  BLOCKNUMBER);
  WHILE (NOT EOF(G))AND(IORESULT=0)
      AND(NOT BADIO)
      AND(BLOCKSTRANSFERRED=1) DO
  BEGIN
    BLOCKSTRANSFERRED:=BLOCKWRITE(F,BUFEER,
                           1,BLOCKNUMBER);
    BADIO:=((BLOCKSTRANSFERRED<1)
               OR(IORESULT<>0));
    BLOCKNUMBER:=BLOCKNUMBER+1;
    BLOCKSTRANSFERRED:=BLOCKREAD(G,BUFFER,
               1,BLOCKNUMBER)
  END;
  CLOSE(F,LOCK)
END.
```

上面的程序从自举盘上读一个名为SOURCE·DATA的磁盘文件，并把它复制到自举盘上另一个叫DESTINAT-

ION的文件中。

12.2.6 预定义文件

标准PASCAL的标准文件INPUT和OUTPUT分别指的是键盘读和显示器输出。此外，APPLE PASCAL还提供了一种KEYBOARD的标准文件。INPUT和KEYBOARD的区别在于：当INPUT用来指键盘时，键入的字符自动显示在屏幕上，而KEYBOARD就不在屏幕上显示输入字符。

INPUT、OUTPUT和KEYBOARD都是交互类型文件，当PASCAL程序开始执行时，它们都是由系统用RESET自动地打开。

12.2.7 输入输出检查

编译选择项可用来控制产生I/O错误检查代码（除UNITREAD、UNITWRITE、BLOCKREAD和BLOCKWRITE外）。默认值为打开状态，即（* $I+*）。在每次调用I/O操作后产生一个I/O检查子程序，如果I/O出错则终止程序执行，显示出错提示信息。

如果关闭I/O检查，即（* $I-*），则不执行运行时间检查。这时I/O操作的结果必须通过调用标准函数IORESULT来代替监视。I/O出错不会造成程序终止，但会终止再一次I/O操作。而当清除错误条件后，调用IORESULT返回0值，程序可适当使用。IORESULT返回0值表示操作成功，其它值表示I/O操作出错（下面介绍IORESULT函数时列出了I/O可能产生的出错信息）。

在发生I/O错而又不希望中断程序执行的情况下，使用

IORESULT函数是非常方便的。

例9　连续请求一文件名，直到成功地打开(即RESET)这个文件（即直到找到该文件）为止。

```
PROCEDURE OPENINFILE;
 BEGIN
  REPEAT
    WRITE('ENTER NAME OF INPUT FILE');
    READLN(INFILENAME);
    (*$I-*)  (*关闭I/O检查*)
    RESET(INFILE,INFILENAME);
    (*$I+*)  (*恢复I/O检查*)
     OK:=(IORESULT=0);
    IF NOT OK THEN
     WRITE('CAN NOT FIND FILE',INFILENAME);
    UNTIL  OK;
 END;
```

下面介绍IORESULT函数。

格式：IORESULT

该函数回送一个整数值，它反映了上一次完成的I/O操作的状态。

下面列出IORESULT可能回送的值

0　无错，完成正常I/O。

1　未用。

2　坏设备卷号。

3　非法操作（例如从PRINTER：读）。

4　未用。

5　设备丢失——没有联机。

6　文件丢失——文件不在目录中。

7　标题错——非法文件名。

8　无空间——盘上没有足够的空间。

9　无设备——设备未安装。

10　在指定的卷上没有这样的文件。

11　复制文件标题。

12　试图打开一个已打开的文件。

13　试图访问一个关闭了的文件。

14　错误的输入格式——读实数或整数数据时出错。

15　循环缓冲区溢出——输入到达太快。

16　写保护错。

64　设备错——磁盘上错误的地址和数据。

12.3　指针类型

1.静态变量和动态变量

前面各章所讨论的量都是静态变量。静态变量必须用VAR说明。VAR说明确定了变量的类型和标识符。标识符可以用来引用变量。以这种方式说明的变量与所在程序体有相同的生命期。若一个程序的存储情况在写程序时就能预计，则可以使用静态变量。

动态变量是应可执行语句的要求而建立的。动态变量不在VAR中说明。因此，动态变量不能用标识符直接引用。对于类型为T的每一个动态变量，都有一个关联的类型为∧T（指向T的指针）的值，这个值用来访问相应的动态变量，并且这个值就是新建立变量的存储地址。

一个动态变量只有当它明显地被破坏或者计算终止时才不存在。

由于动态变量可以属于任何类型T，因此，可以动态地

建立记录变量。如果这些记录包含一个或多个∧T类型的域，那么可以通过把记录连接在一起的方法来建立复杂的结构，参见图12.1。

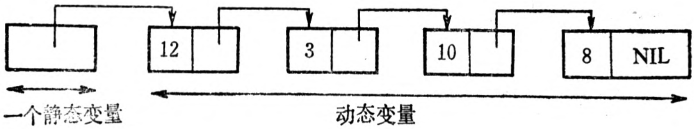


图12.1 存储在动态变量中的第一个序列可把动态变量组织在链中，以建立能用来构造序列的线性表。

2.动态变量的建立

动态变量是通过调用标准过程NEW来建立的。过程NEW的简单调用形式是NEW（P），其中参数P是类型为∧T的指针型变量。该过程的作用是建立一个类型为T的动态变量，在称为“堆”的PASCAL存储区中为该动态变量分配空间，并且也给参数P赋予类型为∧T的一个值，使得可以访问动态变量。

指针类型的格式是：

∧类型标识符

例如：下面的说明适合生成图12.1中的表。

```
TYPE  PTR=^ITEM;
    ITEM=RECORD
    VAL:INTEGER;
    NEXT:PTR
    END;
VAR P,LIST:PTR;
    X:INTEGER;
   INFILE:FILE OF INTEGER;
    .
    .
    .
    NEW(P)
```

这里类型PTR定义为一个指针类型，它指向类型ITEM的变量。这里的变量P、LIST是指针变量，指向类型为ITEM的记录。

3.使用指针

在PASCAL中，指针可按下列方式使用：

被赋值

作为参数

作等与不等的检查

用来访问动态变量

每一种指针类型都有一组值，其中包括值NIL（空）。NIL不指向动态变量，但它可以把值赋给一个指针变量。例如：

P:=NIL;

是正确的。它表示一个空指针，常作链的终点。它在图12.1中用来指明表的结束。

下面以实例来阐述动态变量的访问。前面P和LIST都是静态变量，它们指向类型为ITEM的动态变量。赋值语句：

P:=LIST;

使得P的指针值与LIST相同。记号P∧表示“由P引用的动态变量”，因此P∧为ITEM类型。于是：

P∧:=LIST∧;

表示把LIST所指的动态变量的值赋给由P所指的动态变量。由于P∧为ITEM类型，所以，可使用域选择器。例如：

P∧.VAL:=X;

下面给出一个指针应用的例子。

例10　利用指针进行排序。

```
PROGRAM POINTERDEMO(OUTPUT,INTFILE,NEWFILE);
TYPE  PTR=^ITEM;
      ITEM=RECORD
            VAL:INTEGER;
            NEXT:PTR
            END;
VAR X:INTEGER;
    P,LIST:PTR;
    INTFILE,NEWFILE:FILE OF INTEGER;
BEGIN
  RESET(INTFILE,'WORKDISK:LIST1.DATA');
  REWRITE(NEWFILE,'WORKDISK:LIST2.DATA');
  WHILE NOT EOF(INTFILE) DO
   BEGIN
      (*读入一个序列*)
      LIST:=NIL;  (*表的初始化*)
      READ(INTFILE,X);
      WHILE X<>0 DO
        BEGIN
          NEW(P);
          P^.VAL:=X; (*记下此整数*)
          P^.NEXT:=LIST; (*并连接到表中*)
          LIST:=P;
          READ(INTFILE,X)
        END;
     (*以相反的顺序输出此序列*)
     P:=LIST;
     WHILE P<>NIL DO
       BEGIN
         WRITE(NEWFILE,P^.VAL);
         P:=P^.NEXT  (*推动链向下*)
       END;
     WRITE(NEWFILE,0)
  END
END.
```

4.重新使用动态分配的存储

上面的程序有一个缺点，它没有重新使用不再需要的动态变量。其结果是：程序需要一个足够大的堆来存放整个输入文件与全部指针。一旦堆中内容已经输出，那么它所占的空间应该能够另作它用。如果做到了这一点，堆的容量只要足以存放最长的序列就够了。

可以使用系统提供的几个过程来处理这个问题。

（1）使用MARK和RELEASE过程

在 PASCAL中没有提供标准PASCAL 的 DISPOSE 过程，但提供了MARK和RELEASE过程，用来归还动态分配给系统的存储单元。

格式：MARK（HEAPPTR）；

RELEASE（HEAPPTR）；

这里HEAPPTR具有类型∧INTEGER，在MARK过程中是由引用来调用。MARK置HEAPPTR为系统当前的堆顶指针值。RELEASE置系统的堆顶指针为 HEAPPTR 的值。

下面描述的恢复内存空间的过程大体和过程DISPOSE的功能相似。因为还不能直接要求系统释放被某个特殊变量所占用的空间来作别的用途。

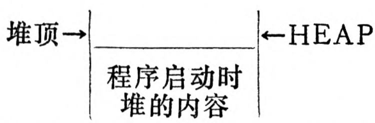
由标准过程NEW所建立的变量被存放在一种“堆”格式结构中。下面以一个程序来说明如何用MARK和RELEASE来改变堆的大小。

例11　释放内存空间的程序例子。

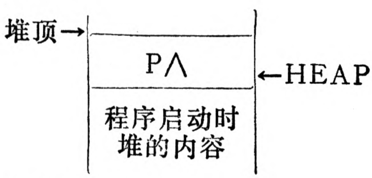
```
PROGRAM SMALLHEAP;
TYPE
   PERSON=RECORD
```

```
            NAME:PACKED ARRAY [0..15] OF CHAR;
            ID:INTEGER
            END;
VAR   P:^PERSON;
      HEAP:^INTEGER;
  BEGIN
    MARK(HEAP);   NEW(P);
    P^.NAME:='FARKLE,HENRY J.';
    P^.ID=999;
    RELEASE(HEAP)
  END.
```

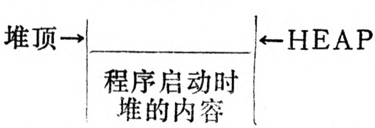
该程序说明了一种特意访问内存内容的轻巧方法（没有别的方法可访问内存内容），它调用MARK把当前堆顶的地址置于度量HEAP中。程序执行到这点时堆内容如下图所示：



接着程序调用标准过程NEW，产生一个新变量P∧，它位于堆中，如下图所示：



调用RELEASE之后，堆变为：



一旦程序不再需要变量P∧，希望释放其内存空间以作

它用，便调用RELEASE，它重置堆顶到变量HEAP所包含的地址。

如果在调用MARK和RELEASE之间，NEW被调用多次，则由这些变量所占用的内存空间立即全部释放。由于堆具有栈的性质，要在堆的中间释放一个单独项所占用的内存空间是不可能的。粗心地使用MARK和RELEASE，可能留下悬空指针，它不再指向由堆空间所定义的内存区域。

（2）MEMAVAIL函数

本函数回送当前栈顶和堆顶之间的存储字数，这就是当前可利用的存储量。

格式：MEMAVAIL

利用这个函数，可以检测栈顶和堆顶之间可用的存储空间量。

习题十二

1.假定类型SEX=（FEMALE，MALE）。定义一个类型STUDENT，它把MALESTUDENT和FEMALESTUDENT两者结合起来。

2.编写一个计算几何图形周长的函数，其首部FUNCTION PERIMETER（FIG：GEOMETRICFIGURE）：REAL；

3.建立家庭成员情况表。它包括每个人的工作证号、姓名、性别、年龄和学龄等。最后计算一下家庭成员的平均年龄、平均学龄。

4.APPLE PASCAL支持几种类型的文件？各种文件变量怎样定义？

5.编写一个程序，计算文本文件中行结束标志的数目。

6.计算文本文件行长度的平均值、最大值和最小值，行结束标志不能看作字符。

7.编写一个程序，把两个有序的INTEGER文件合并起来，产生

一个有序文件。

8.分别叙述∧T、P与P∧等符号的含义。

9.输入一系列整数，当遇到0时停止，建立并输出偶数链表和按从小到大排列的奇数链表。

10.键盘输入20个学生的情况。用指针类型在磁盘中建立学生情况表的文件S1.DATA，包括学生姓名、年龄、性别和成绩。

11.已知一个链表有10个结点A1，A2，…，A_{10}分别存放χ_1，χ_2，…，χ_{10}等10个整型数。要求用整数Y_2代替χ_2，Y_4和Y_5插入A_4和A_5结点之间，并删除结点A_7，形成10个结点的链表。编一程序完成上述功能。

第十三章　过程和函数

使用逐步求精的程序设计方法，一个大的 PASCAL 程序可以由一个或多个模块构成，每个模块还可以由子模块构成。而过程和函数都可以用于构成这种模块。过程和函数一旦定义，便可在程序的语句部分调用。

过程和函数有预定义的或用户定义的。预定义的过程、函数分别称为标准过程和标准函数。这些标准过程和函数是PASCAL的组成部分，无需说明，可直接调用。

13.1　过程的定义和调用

1. 过程的建立方式

过程是一个题目中某个子任务的程序，它在过程说明部分定义。其格式如下：

PROCEDURE 〈过程名〉(〈形参表〉)；

〈程序体〉

过程的格式和程序的格式相同。形参表和说明部分中说明的所有标识符是该过程的局部变量，称为标识符的作用域，在作用域外就失去了其作用。过程说明可以引用外部过程定义的任何常数、类型、变量、过程或函数。过程允许嵌套，即在过程体中使用了过程标识符本身，这种情况称为递归调用。过程一旦在说明部分说明之后，便可以在程序的语句部分用过程名来引用，即过程语句。

下面是定义和调用过程的一个例子。

例1　将两数相加并且整齐地打印出来。

```
PROGRAM  ADDNUMBER(INPUT, OUTPUT);
VAR
   NUM1, NUM2, TOTAL: REAL;
PROCEDURE  DRAWALINE;
   CONST  LENGTH=10;
   VAR  I: INTEGER;
   BEGIN
     FOR  I:=1  TO  LENGTH  DO
        WRITE('-');
   WRITELN
   END;
BEGIN
  READ(NUM1, NUM2);
  WRITELN(NUM1:10:3);（变量NUM1占10位，小数点
                                    后占3位。）
  WRITELN(NUM2:10:3);
  DRAWALINE;
  TOTAL:=NUM1+NUM2;
  WRITELN(TOTAL:10:3)
  END.
```

2.参数

参数是过程内部使用的变量。过程定义时，过程的所有参数的类型都是在过程首部的圆括号内说明，这种参数称为形式参数，或形参；过程调用时，对应每个形参都要给出实在参数值，称为实参。

PASCAL语言支持两种不同的参数，即值参数(用以改变动作）和变量参数(以便取得结果)。上述的例子中，如果

要求过程DRAWALINE在每次调用时输出不同长度的直线，那么LENGTH便可以作为值参，如例2所示。

例2 具有值参的过程例子。

```
PROGRAM VALPARA(INPUT.OUTPUT);
VAR X,Y,N:INTEGER;
   NUMBER:REAL;
PROCEDURE  DRAWALINE(LENGTH:INTEGER);
  VAR I:INTEGER;
  BEGIN
  FOR I:=1 TO LENGTH DO
      WRITE('-');
  WRITELN.
  END;
 BEGIN
  READ(N);
  FOR X:=1 TO N DO
  BEGIN
   READ(NUMBER);
   Y:=ROUND(NUMBER);
   IF Y<0 THEN  DRAWALINE(0)
   ELSE IF Y>100 THEN DRAWALINE(100)
      ELSE  DRAWALINE(Y)
  END
 END.
```

动作的集合构成自包含的过程，此过程能产生后面程序中将要用到的结果。这是通过使用“变量”参数来达到的。在过程调用时，对应于形式变量参数的实在变量参数必须用变量的形式给出。在过程的首部，任何形式变量参数表都是通过在它们的前面冠以保留字VAR来加以区分的。下面的程序中，过程SWAP把实在变量参数X和Y的值进行了互换。

例3 将N组数据对中的每一组进行排序。

```
PROGRAM VARPARA(INPUT,OUTPUT);
VAR I,N:INTEGER;
    X,Y:REAL;
PROCEDURE  SWAP(VAR P,Q:REAL);
  VAR TEMP:REAL;
  BEGIN
   TEMP:=P;
    P:=Q;
    Q:=TEMP
  END;
BEGIN
  READ(N);
  FOR I:=1 TO N DO
    BEGIN
     READ(X,Y);
     IF X>Y THEN SWAP(X,Y);
     WRITELN(X,Y)
    END;
  WRITELN('ARE THE ORDERED PAIRS')
END.
```

在一个过程中，对形式变量参数(例如，P和Q)的任何操作实际上是对相应的实在参数(例如X和Y)进行的。因而上例中，通过过程调用而互换了X和Y的值。相反，形式值参数可以改变其值而不影响过程外的任何变量。

下例表明了一个过程中可以同时使用值参数和变量参数。

例4 将英吋转换为英里和码。

```
PROGRAM VALVARPARA(INPUT,OUTPUT);
VAR A,B,NUMBER:INTEGER;
PROCEDURE  CONVERT(VAR X,Y:INTEGER;
     INS:INTEGER);
  BEGIN
   X:=INS DIV(1760*36);
```

```
  INS=INS MOD(1760*36);
  Y:=INS DIV 36
END;
BEGIN
  READ(NUMBER);
  CONVERT(A,B,NUMBER);
  WRITELN(A:4,'MILES',B:4,'YARDS')
END.
```

在该程序的过程调用中，值参数INS由READ语句所读的值给出。在过程内对两个形式变量参数X和Y的操作实际上是通过A和B来实现的，然后把它们打印出来。

变量参数和值参数不限于四种标准的纯量类型。然而，构造类型的参数通常只能是变量参数（例如：集合、数组和记录），这是因为值参数隐含着一个局部的副本。此外，文件参数必须说明为VAR，而紧缩结构的成份不能作实在变量参数。

PASCAL不允许用过程名或函数名作为过程或函数的形参。

13.2 函数的定义和调用

过程可以用来标识一组操作动作，而"函数"则可以用来得到一个值。函数一经定义，就可以作为表达式的一个项而参与相应的运算。正如任何其它值一样，由函数计算出来的值的类型必须与函数所在表达式类型相一致。函数的类型（不能为构造类型）在函数说明中给出。函数说明格式如下：

FUNCTION 〈函数名〉(〈形参表〉)：〈类型标识符〉；

〈程序体〉

在函数说明的程序体中，可以包含任何语句，而且必须至少有一个用于对函数名赋值的语句被执行。这个语句的作用是返回函数的值。

例5　求X、Y两个实数中哪个最大？

```
FUNCTION MAX(X,Y:REAL):REAL;
 BEGIN
   IF X>Y THEN MAX:=X
   ELSE MAX:=Y
 END;
```

在函数的程序体中，可以调用函数自身，这称为逆归函数。

例6　求N的阶乘。

```
PROGRAM FACTORIAL(INPUT,OUTPUT);
VAR NUMBER:INTEGER;
FUNCTION  FACTORIALFUN(VALUE:INTEGER):REAL;
BEGIN
 IF VALUE=0 THEN  FACTORIALFUN:=1  (*0!=1*)
 ELSE  FACTORIALFUN:=VALUE*
           FACTORIALFUN(VALUE-1);
END;  (*N!=N*(N-1)*)
BEGIN
 READLN(NUMBER);
 IF NUMBER<0 THEN WRITELN('NOT VALID INPUT!')
 ELSE WRITELN(NUMBER,'!=',
        FACTORIALFUN(NUMBER):8:0)
END.
```

13.3　标准过程和标准函数

PASCAL提供了如下三类标准过程和标准 函数：

1.串标准过程和函数

包括：LENGTH函数，POS函数，CONCAT函数，COPY函数，DELETE过程，INSERT过程和STR过程，共7个。

2.文件处理标准过程和函数

它们是：REWRITE过程，RESET过程，CLOSE过程，EOF函数，EOLN函数，GET过程，PUT过程，IORESULT函数，READ过程，READLN过程，WRITE过程，WRITELN过程，PAKE过程，SEEK过程，BLOCKREAD函数和BLOCKWRITE函数，共16个。

3.指针处理标准过程和函数

它们是：MARK过程，RELEASE过程和MEMAVAIL函数，共3个。

上述三类过程和函数，前面已经介绍了。除此之外，还有如下一些标准过程和函数。

13.3.1 处理字节的过程和函数

面向字节的标准过程和标准函数不对参数进行任何范围检查，也不对参数SOURCE和DESTINATION进行类型检查，所以系统并不保护自己免受它们的破坏。使用这些过程和函数前要特别注意。

1.SIZEOF函数

格式：SIZEOF（IDENTIFIER）

IDENTIFIER为变量标识符或类型标识符。

该函数回送一个整数值，这个值为变量IDENTIFIER或类型为IDENTIFIER的任何变量所占用的字节数。

SIZEOF对过程FILLCHAR，MOVELEFT和MOV-

ERIGHT特别有用（见下面）。

2.SCAN函数

格式：SCAN（LIMIT，PEXPR，SOURCE）

该函数扫描一个内存字节区，寻找一个指定的目标字符（紧缩情形，一个字符占用一个字节，等价于寻找一个指定的字节）或与一个指定字符不相等的字符。执行该函数后回送一个整数值，它表示被扫描的字节个数。其中：

LIMIT——整型值，它给出由指定的起点位置开始扫描的最大字节数。若LIMIT为负，则从起点向后回扫。若没有找到目标，则回送LIMIT值。

PEXPR——指明要扫描的那个目标，取下列形式之一：

=CH　　目标是与CH相等的字符。

< >CH　　目标是与CH不相等的字符。

CH可以是产生字符类型结果的任何表达式。

SOURCE——除文件类型外的任何类型变量，它的首字节是扫描的起点位置。

每当找到目标或已经扫描了LIMIT个字节时，SCAN终止执行并回送所扫描的字节个数。如果目标就在起始点则回送值0；如果LIMIT是负值，则向后回扫，回送的值也将是负的。

假设DEM说明为：

VAR DEM：PACKED ARRAY [0..100] OF CHAR；

又设DEM的前面53个元素装有字符：

……THE PTERO IS A MEMBER OF THE PTERODACTYL FAMILY

SCAN(-26, =':', DEM〔30〕); 则回送-26。

SCAN(100, < >'.', DEM); 则回送5。

SCAN(15, =' ', DEM〔5〕); 则回送3。

3.MOVELEFT和MOVERIGHT过程

格式：MOVELEFT(SOURCE,DESTINATION COUNT)。

MOVERIGHT(SOURCE,DESTINATION COUNT);

SOURCE和DESTINATION是除文件类型外的任何类型变量。SOURCE的首字节是要传递的字节区域的起始位置，DESTINATION的首字节是要复制进入的字节区域的起始位置。COUNT 为整型值，指明要传送的字节个数。

过程MOVELEFT将SOURCE字节区域左端开始的COUNT个字节，从左向右依次传送到DESTINATION字节区域左端开始的COUNT个字节位置。

过程MOVERIGHT将SOURCE字节区域右端开始的COUNT个字节，从右向左依次传送到DESTINATION字节区域右端开始的COUNT个字节位置。

提供这两种传送方式是考虑到SOURCE和DESTINATION的字节区域可能重叠。如果重叠，字节传送的方向是关键的，每个字节必须在它被另外的字节覆盖以前移走。

当SOURCE和DESTINATION均为同一个PACKED ARRAY OF CHAR的子数组时，若DESTINATION具有高于SOURCE的下标，则须使用MOVERIGHT，反之使用MOVELEFT。

例如，设变量A说明为：

VAR A：PACKED ARRAY〔1..30〕OF CHAR；

又设数组A中装有字符：

THIS IS THE TEXT IN THIS ARRAY

调用MOVERIGHT（A〔10〕，A〔1〕，10）后A中内容变为：

NE TEXT IN THIS ARRAY

再调用MOVELEFT（A〔1〕，A〔3〕，10）后：

NENENENENENETEXT IN THIS ARRAY

4.FILLCHAR过程

格式：FILLCHAR（DESTINATION，COUNT，CHARACTER）；

DESTINATION是除文件类型之外的任何类型变量，COUNT为整型值，CHARACTER为字符值。

该过程将字符值CHARACTER填入到DESTINATION的首字节开始的COUNT个字节区中。

13.3.2 与设备有关的过程和函数

下面的标准过程和标准函数是面向设备的低级I/O过程和函数，使用时要特别小心。

1.UNITREAD和UNITWRITE过程

格式：UNITREAD（UNITNUMBER，ARRAY，LENGTH〔，〔BLOCKNUMBER〕〔，MODE〕〕）；

UNITWRITE（UNITNUMBER，ARRAY，LENGTH〔，〔BLOCKNUMBER〕〔，MODE〕〕）；

其中UNITNUMBER为I/O设备的卷号，整型数。ARRAY为紧缩数组名，可置上下标以指明开始位置，用作传送的起始地址，亦可用串来表示，但其下标应大于0，因串的0

号元素包含通常不传送的数据。LENGTH取整型值，用来指出传送的字节数。BLOCKNUMBER取整数，仅当使用磁盘驱动器时才有意义，它是传送开始处的绝对块号，如果缺省它，默认为0。MODE取0..15中的整数，默认为0，它控制两种UNITWRITE选择。MODE对UNITREAD没有作用。

由参数MODE控制的UNITWRITE选择仅用于象控制台或打印机这样的面向文本的I/O设备，不能用于磁盘。若参数MODE缺省，对下面两种选择均有效。

一种选择是DLE码的转换。在PASCAL磁盘文件中，行开始处的前导空格表示为DLE字符（ASCII 16）后跟一数码，这个数码为32加上空格数。在对一个非块结构的设备，如打印机输出时，DLE转换选择查明DLE码并把它转换成一串空格符。若MODE值的第三位为二进制1，则不进行DLE转换。

另一种选择是自动换行。在PASCAL磁盘文件中，每行末尾都标有行结束符↲（ASCII 13），而不需别的换行字符。对一个非块结构的设备如打印机，自动在每个↲字符后面插入一个LF字符（ASCII 10）。若MODE的值第二位为二进制1，则不进行自动换行。

MODE值仅第二和第三位有意义，第二位本身对应值4，第三位对应值8。下面列出用来控制选择的MODE值。

MODE＝0　默认值，使两种选择均有效。

MODE＝4　自动换行无效，DLE转换有效。

MODE＝8　DLE转换无效，自动换行有效。

MODE＝12 两种选择均无效。

2.UNITBUSY函数

回送一个布尔值，表明一个指定的设备是否忙闲。如果值为真，表示该指定的设备正等待完成一个I/O传送。UNITNUMBER为指定的设备号。

由于中华学习机的I/O驱动器不能中断驱动，故UNITBUSY总是假值。

3.UNITWAIT过程

等待指定的设备完成正在进行的I/O。因中华学习机驱动器I/O不能被中断驱动，UNITWAIT什么也不做。

4.UNITCLEAR过程

对指定的设备撤消所有I/O操作，并重置加电状态。

格式：UNITCLEAR(UNITNUMBER);

UNITNUMBER为指定的设备号。如果该指定的设备不存在，IORESULT被置为非0值，因此可利用它来检测所给的设备是否在系统中。

13.3.3 超越函数

在PASCAL中，超越函数是作为程序包由系统库来提供的，要想使用这组函数，应在程序首部之后放上说明：

USES TRANSCENT;

这些函数有：

SIN(ANGLE)	正弦
COS(ANGLE)	余弦
EXP(NUMBER)	指数函数
ATAN(NUMBER)	反正切
LN(NUMBER)	自然对数
LOG(NUMBER)	10为底的对数

SQRT(NUMBER)　　　　　　平方根

其中ANGLE和NUMBER均是实数，且变元ANGLE是以弧度表示的。除函数ATAN回送的是弧度值外，其余6个函数均回送实数值。

例7　计算函数

$$F(x)=\begin{cases} e^x & x<0 \\ \dfrac{\sqrt{x}}{1+\cos x} & x\geqslant 0 \end{cases}$$

```
PROGRAM TRANSCENDEMO(INPUT, OUTPUT);
USES TRANSCEND;
VAR  X, F:REAL;
BEGIN
  READ(X);
  IF X<0 THEN F:=EXP(X)
  ELSE F:=SQRT(X)/(1+COS(X));
  WRITE('X=', X, 'F=', F)
END.
```

13.3.4　其它过程和函数

1.TRUNC函数

格式：TRUNC(NUMBER)

其中NUMBER为实数或长整数。如果NUMBER的绝对值超过MAXINT将发生溢出。如果NUMBER为实数，TRUNC回送NUMBER的整数部分；如果NUMBER为长整数，TRUNC回送一个与NUMBER数值相等的整数值。

2.PWROFTEN函数

格式：PWROFTEN(EXPONENT)

这里EXPONENT是0..37中的任何整数值。函数回送10的EXPONENT次幂的值。

3.HALT过程

格式：HALT；

该过程生成一个HALT操作码，执行时引起一个非致命性的运行错。

4.EXIT过程

格式：EXIT（PROCEDURENAME）；

EXIT（PROGRAM）；

EXIT（PROGRAMNAME）；

该过程终止程序、函数或过程的执行，按顺序退出函数、过程或程序本身。

PROCEDURENAME是要退出的过程名和函数名。该过程或函数不一定要是EXIT语句所在的过程和函数。EXIT跟踪过程或函数调用，退回到所指定的过程或函数。跟踪回溯过程中的每一个过程或函数都被退出。若指定的过程是递归的，则最近的一次过程调用被退出。当一个过程或函数经EXIT退出时，任何用于该过程或函数中的文件均自动关闭。

用EXIT退出一个函数时，若在EXIT执行前函数标识符未被赋值，则函数回送一个不确定的值。

当用程序名或PROGRAM作EXIT的参数时，就按指定的程序来终止。

5.GOTOXY过程

格式：GOTOXY（XCOORD，YCOORD）；

该过程置光标到屏幕的指定位置。其中XCOORD和YCOORD取整数值，表示X（水平）和Y（竖直）坐标。

XCOORD必须取0..79的数；而YCOORD为0..23。屏幕左上角的坐标为（0，0）。过程调用后光标处于坐标（XCOORD，YCOORD）处。

6.TREESEARCH函数

格式：TREESEARCH（ROOTPTR，NODEPTR，NAME）

ROOTPTR是指向被搜索的树根结点的指针。NODEPTR是被该函数修改的指针变量。NAME是一个紧缩字符数组（PACKED ARRAY〔1..8〕OF CHAR）标识符，它表示树中被搜索的由8个字符组成的名字。

二元树的结点设为：

```
NODE=RECORD
  NAME: PACKED ARRAY [1..8] OF CHAR;
  LEFTLINE, RIGHTLINE: ∧NODE;
  :(其它的任何域)
  END;
```

的链记录。

类型名和域名并不重要，仅假定记录的头8个字节包含一个8字符名字，跟在后面的两个指针指向其它的节点。

假定树中没有重名并按字母顺序规则赋给结点，对一给定的结点，其左子结点的名字按字母顺序小于结点名字，而右子结点名字大于结点名字。最后，不指向任何其它结点的链是NIL。

TREESEARCH回送下面三个值之一：

0 送给该函数的名字已在树中找到，NODEPTR正指向它。

1 NAME不在树中。如果把它加到树中，应该是NODEPTR所指向的结点的右子结点。

-1 NAME不在树中。如果把它加到树中，应该是NODEPTR所指向的左子结点。

该函数对参数不做任何类型检查。

习 题 十 三

1.编写一个函数MIN，它给出两个数的最小值。在一个程序中利用函数MIN，使该程序打印出一组读入数的最小值和平均值。

2.编写一个完成"时间"加法的程序，时间以年数、周数和小时数给出。解法应包含过程和函数，可设一年正好365天。

3.给出下述程序运行后的结果。

```
PROGRAM NONSENSE(OUTPUT);
VAR THING:INTEGER;
PROCEDURE  CHEAT(VAR HEE,HAW:INTEGER);
   BEGIN
     HEE:=-1;
     HAW:=-HEE
   END;
   BEGIN
     THING:=1;
     CHEAT(THING,THING);
     WRITELN(THING)
   END.
```

```
PROGRAM RUBBISH(OUTPUT);
VAR THING:INTEGER;
PROCEDURE  LIAR(VAR HEE:INTEGER;HAW:INTEGER);
  BEGIN
   HEE:=10*HAW
  END;
  BEGIN
   THING:=10;
   LIAR(THING,THING);
   WRITELN(THING)
  END.
```

4.写一个函数DIGIT(N, K)，它回送数N的从右边开始的第K个数字的值。例如：

DIGIT(254693, 2)=9

DIGIT(7622, 6)=0

5.写一个函数，求两个数的最大公因数。

附录一 FORTRAN语言一览表

1.1 FORTRAN语句一览表

语句名称	格式	功能
标号赋值语句	ASSIGN〈标号〉TO〈变量〉	将〈标号〉赋给〈变量〉
回退语句	BACKSPACE〈部件号〉 或 BACKSPACE(〈变量〉)	将文件指针退回一个记录并，指向记录的始点。
字符赋值语句	CHR=〈表达式〉	将字符表达式的值赋给字符变量。
继续语句	CONTINUE	作循环的终端语句或其它用。
调用语句	CALL〈子例程名〉(〈实参表〉)	调用子例程子程序。
关闭语句	CLOSE(〈文件信息表〉)	关闭已打开的文件。
字符型说明语句	CHARACTER〈变量表〉 或 CHARACTER*n〈变量表〉	说明字符变量。

语句名称	格　　式	功　　能
公用语句	COMMON〈变量表〉或 COMMON/〈块名〉/〈变量表〉	说明有名或无名公用块变量。
循环语句	DO〈标号〉I=〈循参表〉	指明一个循环。
维说明语句	DIMENSION〈数组说符表〉	说明数组的维数、维界等。
数据语句	DATA〈变量表〉/〈数据表〉/……	对变量赋初值。
ELSEIF语句	ELSEIF (〈表达式〉) THEN	多路判定结构中使用。
ELSE语句	ELSE	判定结构中使用。
ENDIF语句	ENDIF	判定结构结束用。
文件结束语句	ENDFILE(〈整数〉)或 ENDFILE(〈变量〉)	写文件结束记录。
程序单位结束语句	END	结束一个程序单位。
等价语句	EQUIVALENCE〈等价域表〉	建立等价关系。
外部语句	EXTERNAL〈外部程序名表〉	说明外部程序单位。
函数子程序语句	FUNCTION〈程序名〉(〈形参表〉) 或〔TYPE〕FUNCTION〈程序名〉(〔〈形参表〉〕)	定义函数子程序、形参等。
格式语句	FORMAT(〈格式编辑符串〉)	为输入输出语句提供用户格式。
语句函数语句	F(〈形参表〉)=〈表达式〉	定义程序单位内的函数关系。
无条件转移语句	GOTO〈标号〉	控制转移到指定的语句去执行。

语句名称	格　　式	功　　能
计算转移语句	GOTO(〈标号表〉)〈表达式〉	根据表达式的值转指定的语句去执行。表达式值指定标号在标号表中的位置。
赋值转移语句	GOTO〈表达式〉(〈标号表〉)	控制转移。表达式由ASSIGN赋值。
算术条件语句	IF(〈算术表达式〉)〈标号表〉	根据算术表达式的值控制转移。
逻辑条件语句	IF(〈逻辑表达式〉)〈语句〉	根据逻辑表达式的值执行其后的〈语句〉。
块IF语句	IF(表达式)THEN	判定结构的开始语句。
整形变量说明语句	INTEGER〈变量表〉	定义整形变量。
隐含类型语句	IMPLICIT〈类型〉(〈字符表〉)	定义隐含规则。
内部语句	INTRINSIC〈内部程序名表〉	指明内部函数。
逻辑赋值语句	LOG=〈表达式〉	将表达式的逻辑值赋给逻辑变量。
逻辑量说明语句	LOGICAL〈变量表〉	定义逻辑变量。
打开语句	OPEN(〈文件说明信息表〉)	打开一个用户文件。
主程序语句	PROGRAM〈主程序名〉	定义主程序名。
暂停语句	PAUSEn或PAUSE	暂停程序的执行。
输入语句	READ(〈文件说明信息表〉)〈对象表〉 或READ(〈部件〉,〈格式〉)〈对象表〉	输入数据。
返绕语句	REWIND〈整数〉或REWIND(〈变量〉)	将文件指针返到文件的开始。
实型量说明语句	REAL〈变量表〉	定义实型变量。

语句名称	格　　式	功　　能
返回语句	RETURN	使程序的执行返回到引用程序单位。
保留定义语句	SAVE〈块名表〉	保留公用块定义。
停语句	STOP n或STOP	停止整个程序的执行。
子例程语句	SUBROUTINE 〈程序名〉(〈形参表〉) 或SUBROUTINE〈程序名〉	子例程单位的开始语句，定义子例程名及其形参。
算术赋值语句	〈变量〉=〈算术表达式〉	将算术表达式值赋给变量。
输出语句	WRITE(〈部件〉,〈格式〉)〈对象表〉或 WRITE(〈文件说明信息表〉)〈对象表〉	输出数据。

1.2　内部函数表

函数名	功　　能	自变量类型	函数类型
AINT(X)	取自变量的整数部分。	实	实
ANINT(X)	取自变量舍入后的实整数部分。	实	实
AMOD(X_1, X_2)	取X1/X2的余数。	实	实
AMAX0(I, J, …)	取最大值。	整	实
AMAX1(X, Y, …)	取最大值。	实	实
AMIN1(X, Y, …)	取最小值。	实	实
AMIN0(I, J, …)	取最小值。	整	整
ALOG(X)	自然对数。	实	实
ALOG10(X)	常用对数。	实	实
ASIN(X)	反正弦。	实	实
ACOS(X)	反余弦。	实	实

函数名	功能	自变量类型	函数类型
ATAN(X)	反正切。	实	实
ATAN2(X, Y)	反正切。	实	实
CHAR(I)	将整型值转换为字符值。	整	字符
COS(X)	余弦。	实	实
COSH(X)	双曲余弦。	实	实
DIM(X, Y)	正差。	实	实
EXP(X)	求e^X值。	实	实
EOF(I)	文件结束。	整	逻辑
FLOAT(I)	将自变量I的值转换为实型值。	整	实
INT(X)	取X的整数部分值。	实	整
IFIX(X)	取X的整数部分值。	实	整
ICHAR(C)	将字符表达式C的值转换为整型值。	字符	整
IABS(X)	取自变量X的绝对假。	实	实
IABS(I)	取自变量I的绝对值。	整	整
ISIGN(I_1, I_2)	符号传送。	整	整
IDIM(I, J)	求正差。	整	整
LGE(C_1, C_2)	按字母顺序比较$C_1 \geq C_2$（大于等于）。	字符	逻辑
LGT(C_1, C_2)	按字母顺序比较$C_1 > C_2$（大于）。	字符	逻辑
LLE(C_1, C_2)	按字母顺序比较$C_1 \leq C_2$（小于等于）。	字符	逻辑
LLT(C_1, C_2)	按字母顺序比较$C_1 < C_2$（小于）。	字符	逻辑
MOD(I_1, I_2)	取I_1/I_2的余数。	整	整
MAX0(I, J, …)	取最大值。	整	整
MAX1(X, Y, …)	取最大假。	实	整
MIN0(I, J, …)	取最小假。	整	整
MIN1(X, Y, …)	取最小值。	实	实
NINT(X)	取自变量舍入后的整数部分。	实	整
REAL(I)	将自变量I的值转换为实型值。	整	实

函数名	功能	自变量类型	函数类型
SIGN(X, Y)	符号传送。	实	实
SQRT(X)	求自变量X的平方根。	实	实
SIN(X)	求自变量X的正弦值。	实	实
SINH(X)	求自变最X的双曲正弦值。	实	实
TAN(X)	求自变量X的正切假。	实	实
TANH(X)	求自变量X的双曲正切假。	实	实

1.3 FORTRAN出错信息

1.编译出错信息

1.读源程序块的致命错误。

2.标号域中出现非数值字符。

3.后续行太多。

4.遇到文件的异常结束。

5.后续行出现标号。

6.在 $编译程序控制命令中遗失了字段。

7.无法打开由 $命令所指明的列表文件。

8.无法识别的 $命令。

9.输入的源文件不符合文本文件格式。

10.所含文件的最大嵌套层数超过了规定值。

11.整型常数溢出。

12.实型常数溢出。

13.常量中的数字太多。

14.标识符太长。

15.字符常量扩展列行末端。

16.字符常量的长度为零。

17.输入中出现非法字符。

18.缺整型常量。

19.缺标号。

20.标号错误。

21.缺类型名（INTEGER，LOGICAL，或CHARACTER〔*n〕）。

22.缺整型常量。

23.语句结束处有多余字符。

24.缺“(”（即缺左括号）。

25.字母被IMPLICIT语句多次说明。

26.缺“)”（即缺右括号）。

27.缺字母。

28.缺标识符。

29.DIMENSION语句中缺少维数。

30.多次指定数组维数。

31.数组维数超过三维。

32.EQUIVALENCE语句中出现不相容的自变量。

33.在一个类型说明语句中同一变量出现多次。

34.该标识符已被说明过。

35.该内部函数不能作为自变量传递。

36.标识符必须是变量。

37.标识符必须是变量或是当前的FUNCTION名。

38.缺“/”（即缺斜杠）。

39.被取名的公用区已保存起来。

40.变量已出现在公用区中。

41.在两个不同的公用区中的变量不可等价。

42.EQUIVALENCE语句中的下标数与该变量的说明不符。

43.EQUIVALENCE语句中下标超界。

44.两个不同单元等价一个公用区中的同一存储单元。

45.EQUIVALENCE语句向负方向扩展公用区。

46.EQUIVALENCE语句使一变量成为两个不同的存储单元（不在公用区中的）。

47.缺语句标号。

48.在同一公用区中，不允许字符和数字的混合项。

49.字符项不能与非字符项等价。

50.在表达式中有非法字符。

51.在表达式中不能使用子例程子程序名。

52.自变量类型必须是整型或实型。

53.自变量类型必须是整型或实型或字符型。

54.比较的类型必须相容。

55.表达式的类型必须是逻辑型的。

56.下标太多。

57.下标太少。

58.缺变量。

59.缺“=”号。

60.等价字符的长度必须相等。

61.非法赋值——类型不匹配。

62.只能调用子例程子程序。

63.在公用区中不能出现形参自变量。

64.在等价语句中不能出现形参自变量。

65.假定大小数组说明仅能用于形参自变量。

66.可调数组说明只能用于形参数组。

67.假定大小数组的维说明符“*”必须是最后一维的上界。

68.可调维界必须是形参自变量或在预先出现的公用区内。

69.可调维界必须是只包含常数、公用区变量或常数名的整型表达式。

70.主程序不能多于一个。

71.有名公用区的大小在所有的过程中必须一致。

72.形参自变量不能出现在DATA语句中。

73.在DATA语句中不得出现无名公用区的变量。

74.在DATA语句中不得出现子例程子程序名、函数子程序名、内部函数名等。

75.DATA语句中的下标越界。

76.循环次数必须是大于0的整数

77.缺常数。

78.DATA语句中的类型混乱。

79.变量的个数与DATA语句表中的值的个数不一致。

80.语句不能有标号。

81.无此内部函数。

82.内部函数的类型说明与内部函数的实际类型不匹配。

83.缺字母。

84.FUNCTION的类型与先前使用的不符。

85.该子例程子程序已在本次编译中出现过。

86.在其它程序单位中已存在的这个过程，已由 $US命令定义。

87.内部函数的自变量类型出错。

88.SUBROUTINE/FUNCTION先前已作为SUBROUTINE/FUNCTION使用过。

89.不可识别的语句。

90.函数不可是字符型。

91.缺END语句。

92.一个程序单位不能出现在 $SEPATE编译中。

93.在FUNCTION或SUBROUTINE调用中，实参个数少于形参个数。

94.在FUNCTION或SUBROUTINE调用中,实参个数多于形参个数。

95.实参与形参类型不匹配。

96.调用未定义的过程。

97.该过程已由 $EXT命令所定义。

98.字符型量的长度为1到225之间。

100.语句顺序错。

101.不可识别的语句。

102.非法转移到块中。

103.标号已用于FORMAT语句。

104.标号已定义。

105.转向FORMAT语句的标号。

106.在此处禁止出现DO语句。

107.DO标号必须在DO语句后面。

108.此处禁止用ENDIF语句。

109.没有与这个ENDIF匹配的IF语句。

110.在IF块中，DO循环的嵌套不当。

111.此处禁止用ELSEIF语句。

112.没有与ELSEIF匹配的IF语句。

113.DO循环或ELSEIF块的嵌套不当。

114.缺“(”号。

115.缺“)”号。

116.缺THEN。

117.缺逻辑表达式。

118.此处禁止用ELSE语句。

119.没有与ELSE匹配的IF语句。

120.此处禁止用无条件GOTO语句。

121.此处禁止用赋值GOTO语句。

122.此处禁止用块IF语句。

123.此处禁止用逻辑IF语句。

124.此处禁止用算术IF语句。

125.缺“,”号。

126.表达式类型错。

127.此处禁止用RETURN语句。

128.此处禁止用STOP语句。

129.此处禁止用END语句。

131.引用未定义的标号。

132.DO循环或块IF无终端语句。

133.此处不允许用FORMAT语句。

134.FORMAT语句的标号已经引用过。

135.FORMAT语句必须有标号。

136.缺标识符。

137.缺整型变量。

138.缺“TO”。

139.缺整型表达式。

140.是赋值GOTO，而不是ASSIGN语句。

141.不可识别的字符常量作为任选项。

142.缺字符常量任选项。

143.部件说明要求整型表达式。

144.在CLOSE语句中，“，”号后要求STATUS任选项。

145.在OPEN语句中字符表达式作文件名。

146.在OPEN语句中必须有“FILE=”选择项。

147.在OPEN语句中，“RECL=”任选项说明了两次。

148.在OPEN语句中，“RECL=”选择项所要求的是整型表达式。

149.在OPEN语句中不可识别的任选项。

150.在OPEN语句中，直接存取文件必须指定“RECL=”任选项。

151.可调数组不允许作为输入输出表中的元素。

152.在隐循环中遇到语句结束，以“(”号开头的表达式不允许作为输入输出表的元素。

153.需要隐循环的控制变量。

154.表达式不允许作为输入列表元素。

155.在语句中出现了两次“REC”=选择项。

156.“REC=”缺整型表达式。

157.“END=”任选项只能出现在READ语句中。

158.“END=”任选项在语句中出现了两次。

159.不可识别的输入输出部件。

160.输入输出语句中有不可识别的格式。

161.在输入输出语句中的“，”后要求任选项。

162.不可识别的输入输出列表元素。

163.未定义的FORMAT语句标号。

164.整型变量用作赋值格式，但没有ASSIGN语句。

165.把一个可执行语句的标号用作格式。

166.赋值格式缺少变量。

167.用作格式的标号定义了一次以上。

169.FUNCTION调用需要“( )”。

200.在读 $USES文件中出错。

201. $USES文件中语法出错。

202. $USES文件中的SUBROUTINE名或FUNTION名已经定义过。

203.函数不能送回字符型值。

204.不能打开 $USES文件。

205. $USES语句太少。

206. $USES文件中，没有该程序单位的·TEXT信息。

207. $USES文件中有非法段。

208. $USES没有这个程序单位。

209. $USES语句中遗失了程序单位名。

210. $USES命令结束处有多余的字符。

211.内部程序单位不可被覆盖。

212. $EXT命令中语法出错。

213.子例程子程序不能有类型说明。

214.在 $EXT命令中FUNCTION名或SUROUTINE名已经被 定义过。

400.代码文件写错误。

401.JTAB中的实体太多。

402.段中的子例程子程序和函数太多。

403.过程太长（代码缓冲区太小）。

404.在系统盘上的高速暂存存储器文件空间不够。

450.高速暂存存储器文件上的读错误。

2.运行出错信息

600.FORMAT语句缺少最后的“)”。

601.在输入中不需要符号。
602.在输入中符号后未跟数字。
603.在输入中缺数字。
604.在格式码B后面缺N或Z。
605.在格式码中有不符合的数字。
606.在格式中重复因子不能为零。
607.在格式中W字段缺整数。
608.在格式中W字段缺正整数。
609.在格式中缺“0”。
610.在格式中d字段缺正整数。
611.在格式中e字段缺整数。
612.在格式中的e字段缺正整数。
613.在A格式中的W字段缺正整数。
614.在格式中的H氏字段不允许在REND语句中出现。
615.在格式中的H氏字段缺重复因子。
616.在格式中的X字段需要重复因子。
617.在格式中的P字段缺重复因子。
618.格式中的整数出现在“+”或“-”的前面。
619.格式中的整数出现在“+”或“-“之后。
620.格式中带符号的重复因子之后缺P格式。
621.格式的最大嵌套层越界。
622.格式中“)”有了重复因子。
623.格式中整数后接了一非法字符“,”。
624.“·”是非法的格式控制符。
625.字符常量不可出现在READ语句的格式中。
626.格式中的字符常量不可重复。
627.格式中的“/”不可重复。
628.格式中的“$”不可重复。
629.BN或BZ格式控制不可重复。
630.企图在未知数设备上完成输入输出。

631.企图在作为非格式化打开的文件上进行格式化输入输出。

632.格式应以“(”号开始。

633.整数读缺I格式。

634.实型读缺F或“E”格式。

635.在格式的实型数读中用了两个“·”字符。

636.读格式化实型数时缺数字。

637.读逻辑型数时缺L格式。

639.读逻辑型数时缺T或F。

640.读逻辑型数时缺A格式描述符。

640.读字符型数时缺A格式描述符。

641.写整型数时缺I格式。

642.F格式中的W字段不大于d+1。

643.E格式中比例因子超出了d字段的范围。

644.写实型数时缺E或F格式。

645.写逻辑型数时缺L格式。

646.写字符型数时缺A格式。

647.企图在格式化打开的文件上进行非格式的输入输出。

648.不能写分块的输出——文件所在的设备的空间不够。

649.不能读分块的输入。

650.格式文本文件错——最后512个字符没有回车。

651.输入整型数时溢出。

652.读入的字节数太多，超出直接存取单位记录。

653.以直接存取单位记录读出的字节数目不正确。

654.企图在非块设备上打开直接存取部件。

655.企图在文件结束记录外的部件上进行外部输入输出。

656.企图在非定位记录号上规定直接存取的部件位置。

657.企图在顺序打开的部件上作直接存取。

658.企图在非块设备上规定直接存取的位置。

659.企图在读的文件结束范围之外规定直接存取部件的位置。

660.企图在与非块设备或非格式文件连接的部件上回退（BACKSP-

ACE）。

661.企图回退顺序的非格式的部件。

662.ASIN或ACOS的自变量超界—｜X｜>1.0。

663.SIN或COS的自变量太大—｜X｜>10^6。

664.企图在内部部件上进行非格式输入输出。

665.企图在内部部件上放置一个以上的记录。

666.企图在内部部件上写比其长度更长的字符。

667.在未知部件上调用EOF函数。

697.整型变量当前未赋以FORMAT格式。

698.在没有"END="任选项的读时遇到文件结束。

699.整型变量没有赋以GOTO语句中的标号。

1000⁺编译程序诊断错误信息——在正确的程序中永远不会出现。

附录二　PASCAL语言一览表

2.1　PASCAL命令一览表

BEGIN　语句1；语句2；…；语句n　END；

定义一个复合语句。

CASE〈选择表达式〉OF

情况标号1：语句1；

情况标号2：语句2；

⋮

情况标号N：语句n；　　　　　　　（n≥1）

END；

定义一个情况语句。

FOR〈变量名〉：=〈表达式1〉TO/DOWNTO〈表达式2〉DO

〈语句〉

变量赋初值（〈表达式1〉），执行一次语句，然后增加（TO）1或减少（DOWNTO）1再执行语句，如此重复，直到变量值超过终

限值(〈表达式2〉)为止。

FUNCTION〈函数名〉(〈参数表〉):类型;

〈程序体〉

函数说明。

GOTO〈语句标号〉

转向语句标号所在的语句去执行。

IF〈条件〉THEN〈语句1〉〔ELSE〈语句2〉〕

当条件成立时执行语句1,否则执行语句2。语句1或语句2执行结束后,顺序执行后继语句。其中〔〕表示待选项。

PROCEDURE〈过程名〉(〈参数表〉);

〈程序体〉

过程说明

REPEAT 语句1;语句2;…;语句n UNTIL〈条件〉

反复执行指定的语句,直到条件满足才停止。

WHILE〈条件〉DO 语句;

当条件为真时,反复执行指定的语句,直到条件为假时停止。

WITH〈变量1〉,〈变量2〉,…,〈变量n〉DO 语句;

打开以变量1至变量n为域名的记录,以便使用这些域名,执行DO后语句。

2.2 PASCAL出错信息表

1.编译程序出错信息表

1	ERROR IN SIMPLE TYPN	简单类型错误
2	IDENTIFER EXPECTED	缺标识符
3	'PROGRAM' EXPECTED	缺PROGRAM
4	')' EXPECTED	缺符号')'
5	';' EXPECTED	缺符号';'
6	ILLEGAL SYMBOL (POSSIBLY MISSING ';' ON LINE ABOVE)	非法符号(可能在上行缺';')

7	ERROR IN PARAMETER LIST	参数表中有错误
8	'OF 'EXPECTED	缺 'OF '
9	'('EXPECTED	缺 '('
10	ERROR IN TYPE	类型说明有错
11	' ' EXPECTED	缺空格
12	' ' EXPECTED	同上
13	'END ' EXPECTED	缺END
14	'; ' EXPECTED	缺 '; '
15	INTEGER EXPECTED	缺整型量
16	'=' EXPECTED	缺 '= '号
17	'BEGIN 'EXPECTED	缺BEGIN
18	ERROR IN DECLARATION PART	在说明部分有错误
19	ERROR IN 〈FIELD-LIST〉	记录的域表有错误
20	'· 'EXPECTED	缺 '· '
21	'* 'EXPECTED	缺 '* '
22	'INTERFACE 'EXPECTED	缺接口部分
23	'IMPLEMENTATIAN 'EXPECTED	缺实现部分
24	'UNIT 'EXPECTED	缺少UNIT
50	ERROR IN CONSTANT	常数有错
51	':= ' EXPECTED	缺少赋值 ':= '号
52	'THEN ' EXPECTED	缺THEN
53	'UNTIN ' EXPECTED	循环语句中缺UNTIL
54	'DO ' EXPECTED	缺 'DO '
55	'TO ' OR 'DOWNTO ' EXPECTED IN FOR STATEMENT	在FOR语句中，缺少TO或DOWNTO
56	'IF ' EXPECTED	缺 'IF '
57	'FILE ' EXPECTED	缺 'FILE '
58	ERROR IN〈FACTOR〉(BAD EXPRESSION)	表达式中有错误项。
59	ERRON IN VARIABLE	变量中有错误
101	IDENTIFER DECLARED TWICE	标识符说明了两次。
102	LOW BOUND EXCEEDS HIGH BOUND	下界超过上界。

103 IDENTIFIER IS NOT OF THE APPROPRIATE CLASS

标识符没有适当的类型

104 UNDECLARED IDENTIFIER 标识符未经说明

105 SIGN NOT ALLOWED 符号不允许

106 NUMBER EXPECTED 缺少数

107 INCOMPATIBLE SUBRANGE TYPES 子界类型不适当

108 FILE NOT ALLOWED HERE 这里不能用文件

109 TYRE MUST NOT BE REAL 不能是实型

110 ⟨TAGFELD⟩ TYPE MUST BE SCALAR OR SUBRANGE

记录的标志域必须是枚举或子界类型

111 INCOMPATIBLE MITH ⟨TAGFIELD⟩ PART 记录的标志域不当

112 INDEX TYPE MUST NOT BE REAL 下标类型不能是实型

113 INDEX TYPE MUST BE A SCALAR OR A SUBRANGE

下标类型必须是枚举或子界类型

114 BASE TYPE MUST NOT BE REAL 基类型不能是实型

115 BASE TYPE MUST BE SCALAR OR A SUBRANGE

基类型必须是枚举或子界类型

1116 ERROR IN TYPE OF STANDARD PROCEDURE PARAMETER 标准过程的参数有错误

117 UNSATISFIED FORWARD REFERENCE 不满足向前引用

18 FORWARD REFERFNCE TYPE IDENTIFIER IN VARIABLE DECLARATION 向前引用类型标识符在变量说明里面

119 RE-SPECIFIED PARAMETERS NOT OK FOR A FORWARD DECLARED PROCEDURE

对于前面已经说明了的过程，重新指定参数是不行的。

120 FUNCTION RESULT TYPE MUST BE SCALAR, SUBRANGE OR POINTER 函数结果类型必须是枚举，子界或指针类型

121 FILE VALUE PARAMETER NOT ALLOWED

不允许文件作为参数

122 A FORWARD DECLARED FUNCTION'S RESULT TYPE CAN'T BE RE-SPECIFIED 前面说明的函数的类型不能重新说明

123 MISSING RESULT TYPE IN FUNCTION DECLARATION

在函数说明中漏掉了函数值的类型

124 F—FORMAT FOR REALS ONLY　F—格式仅对实数

125 ERROR IN TYPE OF STANDARD PROCEDUCE PARAMETER　标准过程参数类型有错误

126 NUMBER OF PARAMETER DOES NOT AGREE WITH DECLARATION　参数的数目与说明不一致

127 ILLEGAL PARAMETER SUBSTITUTION　非法的参数替换。

128 RESULT TYPE DOES NOT AGREE WITH DECLARATCON

结果类型与说明不一致

129 TYPE CONFLICT OF OPERANDS　运算对象类型相冲突

130 EXPRESSION IS NOT OF SET TYPE　表达式不能有集合类型

131 TESTS AN EQUALITY ALLOWED ONLY　只能允许检查相等

132 STRICT INCLUSION NOT ALLOWED　不允许严格包含

133 FILE COMPARISON NOT ALLOWED　不允许文件比较

134 ILLEGAL TYPE OF OPESAND(S)　运算对象类型不合法

135 TYPE OF OPERAND MUST BE BOOLEAN

运算对象类型必须是布尔量

136 SET ELEMENT TYPE MUST BE SCALAR OR SU BRANGE

集合元素类型必须是枚举或子界类型

137 SET ELEMENT TYPE MUST BE COMPATIBLE

集合元素类型不相容

138 TYPE OF VARIABLE IS NOT ARRAY　变量类型不是数组

139 INDEX TYPE IS NOT COMPATIBLE WITH THE DECLARATION　下标类型与说明不相容

140 TYPE OF VARIABLE IS NOT RECORD　变量的类型不是记录

141 TYPE OF VARIABLE MUST BE FILE OR POINTER

变量的类型必须是文件或指针类型

142 ILLEGAL PARAMETER SOLUTION　非法的参数替代

143 ILLEGAL TYPE OF LOOP CONTROL VARIABLE

循环控制变量的类型不合法

144 ILLEGAL TYPE OF EXPRESSION 非法的表达式类型

145 TYPE CONFLICT 类型相冲突

146 ASSIGNMENT OF FILES NOT ALLOWED 不允许文件赋值

147 LABEL TYPE INCOMPATIBLE WITH SELECTING EXPRESSION 标号类型与选择表达式不相容

148 SUBRANGE BOUNDS MUST BE SCALAR 子界的界必须是枚举量

149 INDEX TYPE MUST BE INTEGER 下标类型必须是整型

150 ASSIGNMENT TO STANDARD FUNCTION IS NOT ALLOWED 给标准函数赋值是不允许的

151 ASSIGNMENT TO FORMAL FUNCTION IS NOT ALLOWED 给形式函数赋值是不允许的

152 NO SUCH FIELD IN THIS RECORD 本记录中没有这样的域

153 TYPE ERROR IN READ 读语句中类型出错

154 ACTUAL PARAMETER MUST BE A VARIABLE 实参应该是一个变量

155 CONRTOL VARIABLE CANNOT BE FORMAL OR NON—LOCAL 控制变量不能是形式的或非局部的

156 MULTI DEFINED CASE LABEL 多次定义情况标号

157 TOO MANY CASEE IN CASE STATEMENT 情况语句中的CAES过多

158 NO SUCH VARIANT IN THIS RECORD 遗漏记录的变体说明

159 REAL OR STRING TAG-FIELDS NOT ALLOWED 不允许用实数或串作标志域

160 PREVIOUS DECLARATION WAS NOT FORWARD 预先说明的不是FORWARD

161 AGAIN FORWARD DECLARED 重复向前说明

162 PARAMETER SIZE MUST BE CONSTANT 参数的大小必须是常量

163 MISSING VARIANT IN DECLARATION 在说明中遗漏变体

164 SUBSTITUTION OF STANDARD PROC./FUNC NOT ALLO-

WED 不允许标准过程或函数的替换

165 MULTI DEFINED LABEL 多重定义标号

166 MULTI DECLARED LABEL 多重说明标号

167 UNDECLARED LABEL 未说明标号

168 UNDEFINED LABEL 未定义标号

169 ERROR IN BASE SET 集合的基类型出错

170 VALUE PARAMETER EXPECTED 缺值参数

171 STANDARD FILE WAS RE—DECLARED
标准文件被再次说明

172 UNDECLARED EXTERNAL FILE 未说明外部文件

174 PASCAL FUNCTION OR PROCEDURE EXPECTED
期望PASCAL的过程或函数

175 ACTUAL PARAMETER MAX STRING LENGTH<FORMAL MAX LENGTH 实参最大串长<形参最大长度

182 NESTED UNITS NOT ALLOWED 不允许嵌套UNITS

183 EXTERNAL DECLARATION NOT ALLOWED AT THIS NESTING LEVEL 在这一嵌套层不允许外部说明

184 EXTERNAL DECLARATCON NOT ALLOWED IN INTERFACE SECTION 在接口部分不允许外部说明

185 SEGMENT DECLARATION NOT ALLOWED IN UNIT
在UNIT中不允许有SEGMENT说明

186 LABELS NOT ALLOWED IN INTERFACE SECTION
在接口部分不允许有标号

187 ATTEMPT TO OPEN LIBRARY UNSUCCESSFUL 打开库失败

188 UNIT NOT DECLARED IN PREVIOUS USES DECLARATION
在预先USES说明中未说明UNIT

189 'USES' NOT ALLOWED AT THIS NESTING LEVEL
在这层嵌套级不允许USES

190 UNIT NOT IN LIBRARY UNIT不在库中

191 NO PRIVATE FILES 无私用文件

192 'USES' MUST BE IN INTERFACE SECTION

USES应在接口部分

193 NOT ENOUGH ROOM FOR THIS OPERATION

缺乏运行空间

194 COMMENT MUST APPEAR AT TOP OF PROGRAM

注解应在程度的顶部

195 UNIT NOT IMPORTABLE UNIT不可引入

201 ERROR IN REAL NUMBERS—DIGIT EXPECTED

实数出错一期望数字

202 STRING CONSTANT MUST NOT EXCEED SOURCE LINE

串常量不可超过源行

203 INTEGER CONSTANT EXCEEDS 整型常量越界

204 8 OR 9 IN OCTAL NUMBER 八进制数中不可有8或9

250 TOO MANY SCOPES OF NESTED IDENTIFIERS

标识符的嵌套范围太大

251 TOO MANY NESTED PROCEDURES OR FUNCTIONS

过程或函数嵌套太多

252 TOO MANY FORMARD REFERENCES OF PROCEDURE ENTRIES 过程入口的向前引用太多

253 PROCEDURE TOO LONG 过程太长

254 TOO MANY LONG CONSTANTS IN THIS PROCEDURE

本过程的长常数太多

256 TOO MANY EXTERNAL REFERENCES 外部调用太多

257 TOO MANY EXTERNALS 外部过程或函数太多

258 TOO MANY LOCAT FILES 局部文件太多

259 EXPRESSION TOO COMPLICATED 表达式过于复杂

350 DIVISIAN BY ZERO 除0

301 NO CASE PROVIDED FOR THIS VALUE

对此值未提供情况标号

302 INDEX EXPRESSION OUT OF BOUNDS 下标表达式越界

303 VALUE TO BE ASSIGNED IS OUT OF BOUNDS 赋值越界

304 ELEMENT EXPRESSION OUT OF RANGE 元素表达式越界

350 NO DATA SEGMENT ALLOCATED 无数据SEGMENT

351 SEGMENT USED TWICE SEGMENT使用了两次

352 NO CODE SEGMENT ALLOCATED 无代码SEGMENT

353 NON—INTRINSIC UNIT CALLED FROM INTRINSIC UNIT

在内部UNIT中调用非内部UNIT

354 TOO MANY SEGMENT FOR THE SEGMENT DICTIONARY

SEG目录中的SEGMENT太多

398 IMPLEMENTA TUN RESTRICTION 实现的限制

399 IMPLEMENTATION RESTRICTION

400 ILLEGAL CHARACTER IN TEXT 在文本中出现非法字符

401 UNEXPECLED END OF INPUT 输入不应该结束

402 ERROR IN WRITING CODE FILE, NOT ENOUGH ROOM

输出代码文件出错，空间不够

403 ERROR IN READING INCLUDE FILE 读文件时出错

404 ERROR IN WRITING LISTFILE, NOT ENOUGH ROOM

输出列表文件出错，空间不够

405 CALL NOT ALLOWED IN SEPARATE PROCEDURE

406 INCLUDE FILE NOT LEGAL 包含的文件不合法

407 TOO MANY LIBRARIES 库太多

408 (*$S+*) NEEDED TO COMPILE UNITS

编译UNIT需要用选择项(*$S+*)

500 GENERAL ASSEMBLER ERROR 汇编程序出错

2. 运行错误信息

当一个PASCAL程序或者操作系统的某一部分运行出错时，将以下述形式，给出一个运行错误号或运行错误信息：

EXEC ERR#10

S#1, P#7, I#56

TYPE 〈SPACE〉 TO CONTINUE （敲空格就继续）

或： IO ERROR：VOL NOT FOUND （卷号没有找到）

S#1, P#7, I#56

TPYE 〈SPACE〉 TO CONTINUE （敲空格键则继续）

其中S#表示程序的当前段号，P#表示段号的过程号，I#表示在过程中已经查出的出错字节号。只有当表示SYSTEM·PASCAL文件在自举驱动器内时，才能给出更详细的用户I/O错误信息来。

关于段号、过程号和字节号的描述，可在编译源程序时，由L+（编译列表）选择项给出。

运行出错信息表

错误号	错误信息
0	未定义性质的系统错误。
1	串或子界变量越界（如果使用R—编译程序选择项，不会发现此类错误）。
2	无段：代码文件出错。文件可从磁盘上正常读出，但无有效段。
3	从一个不是事先调用过的过程或者不在“活动”的过程出口。
4	栈溢出：程序所用栈要求的空间已超过用户可使用的存储空间。
5	整型量溢出：整型运算结果大于二进制16位（两个字节）或者长整型运算结果大于十进制36位数，或者将长整型量赋给不能满足数位大小的变量。
6	用0作为除数。
7	无效存储访问。（APPLE机未使用）
8	用户中断：按了中断键CTRL—[illegible]
9	系统I/O错：企图从磁盘上读出操作系统的一个段内容。
10	用户I/O错：用户程序企图实现一个BLOCKREAD,BLOCKWRITE，GET，或RUT时出错。如果SYSTEM.PASCAL文件有效，上述错误会进一步以I/O错误信息给出。
11	不能实现的指令：操作码不能执行。
12	浮点运算错：错误由于实型数的格式，实型数溢出等原因引起。
13	串过长：企图将串存入空间大小不足的目标串中引起。
14	暂停（APPLE机未使用）。
15	坏块（APPLE机未使用本错误码，而用 I/O ERROR#64 来代替）。

表中0号和9号错误为致命性的错误，将引起系统自动“冷启动”，或者要用户自己“冷启动”。其余的错误为非致命的错误，将会引起系统重新初始化，即调用INITIALIZE过程进行“热启动”，一般只要按下空格键即可。

3. I/O出错信息

I/O 出 错 信 息 表

错误号	错 误 信 息
0	无错。
1	磁盘有坏块（APPLE机未使本错误号）。
2	坏设备（卷号）。
3	错误的I/O操作方式（例如，企图从打印机上进行读操作）。
4	硬设备未定义（APPLE 机未使用本错误号）。
5	丢失设备：使用某一设备成功地做过一次操作后，该设备突然不在线上。
6	丢失文件：使用某个文件成功地进行操作后，该文件突然从盘目录上被删去。
7	错误文件名：出现非法文件名(例如文件名的长度超过15个字符)。
8	无空间：磁盘上无足够的存储空间（文件必须存储在连续的磁盘块上）。
9	缺设备：所需的设备未联机。
10	缺文件：所需的文件不在指定的磁盘目录上。
11	重写文件。企图再写一遍已经存在在磁盘上的文件。
12	文件未关闭：企图打开已经打开过的文件。
13	文件未打开：企图访问一个关闭的文件。
14	格式出错：错误发生在整型或实型量的READ过程中（例如，程序需要读入整型量，而你却输入一个字符）。
15	缓冲区溢出：输入字符的速度大于缓冲区能够接受字符的速度。
16	写保护错：指定的磁盘已做写保护，不能对该磁盘作写操作。
64	设备出故障：不能正确地读、写（如磁盘上出现坏块等）。

封面设计：阎欢玲

ISBN 7-5053-1108-5/TP·178 定价：3.80 元

FORTRAN与Pascal语言

電子工業出版社